EVANDRO NUNES REGOLIN

PROGRAMAÇÃO GENÉTICA E ALGORITMOS DE ESTIMAÇÃO DE DISTRIBUIÇÃO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Prof.ª Dr.ª Aurora Trinidad Ramirez Pozo

CURITIBA

2004

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno *Evandro Nunes Regolin*, avaliamos o trabalho intitulado, *"PROGRAMAÇÃO GENÉTICA E ALGORITMOS DE ESTIMAÇÃO DE DISTRIBUIÇÃO"*, cuja defesa foi realizada no dia 25 de junho de 2004, às quatorze horas, no Auditório do Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 25 de junho de 2004.

Prof. Dr. Martin Alejandro Musicante
DINF/UFPR – Presidente

Prof. Dr. André Ponce de Leon F. de Carvalho
USP – Membro Externo

Prof. Dr. André Luiz Pires Guedes
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

RESUMO

Esse trabalho desenvolve um estudo sobre a Programação Genética e os Algoritmos de Estimação de Distribuição, visando integrar os benefícios de ambas as técnicas. Como resultado desse estudo, foi proposta a Programação Automática Bayesiana (PAB).

A PAB é uma ferramenta para a programação automática que usa uma gramática livre de contexto e uma distribuição de probabilidade estimada de um conjunto de soluções promissoras para guiar uma busca por uma solução ótima para um problema. Uma rede Bayesiana é usada para modelar a estimação de distribuição de probabilidade.

De modo a manter a diversidade populacional, uma operação genética que usa um conceito de similaridade entre um indivíduo e a população como critério de ativação é também proposta. Para validar a ferramenta, ela foi aplicada em três problemas, e seu desempenho comparado com a Programação Genética tradicional.

ABSTRACT

This work develops a study on Genetic Programming and Estimation Distribution Algorithm aiming to profit from both technics. As result of this study, it has been proposed the Bayesian Automatic Programming (BAP).

BAP is an automatic programming technic which employs a context free grammar and a probability distribution of a set of promising solutions to guide a search for an optimal solution for a problem. This probability distribution estimated is modeled through a Bayesian network.

To keep the population diversity, it has been proposed a genetic operation employing a metric of similarity between an individual and a population, as criteria of activation. To validate the tool, it has been applied to tree problems and its performance compared with the performance of the traditional Genetic Programming.

CAPÍTULO 1

INTRODUÇÃO

Algoritmos Evolucionários (AEs) têm sido usados com sucesso para solucionar uma ampla variedade de problemas de otimização e busca. Dois dos Algoritmos Evolucionários mais conhecidos são os Algoritmos Genéticos [19] e a Programação Genética [21]. Ambos são baseados nos princípios da seleção e variação. Enquanto a seleção tenta colecionar soluções, chamadas indivíduos, que sejam promissoras na resolução de um determinado problema, a variação modifica ou combina características desses indivíduos com o objetivo de explorar novas regiões do espaço de busca.

Apesar de terem sido usados com sucesso em muitos contextos, os AEs possuem muitos problemas. Entre esses problemas, o de acoplamento [15] é um dos mais graves. Ele aparece quando o processo de variação não é capaz de identificar componentes de um indivíduo que não devem ser separados, pois somente quando unidos eles contribuem para a resolução do problema.

Recentemente, têm sido propostos novos tipos de AE, que usam modelos probabilísticos para descrever as características dos melhores indivíduos. Ele são conhecidos como Algoritmos de Estimação de Distribuição (Estimation of Distribution Algorithms - EDAs) [25].

Os EDAs não se baseiam nos mesmos processos de variação usados em AEs, que basicamente modificam componentes dos indivíduos selecionados. Ao invés, eles induzem distribuições probabilísticas a partir de um conjunto de indivíduos selecionados, das quais são produzidos novos indivíduos. Muitas ferramentas que seguem esse paradigma surgiram recentemente, tais como o Algoritmo de Otimização Bayesiana (Bayesian Optimization Algorithm - BOA) [30], a Evolução de Programa Incremental Probabilístico (Probabilistic Incremental Program Evolution - PIPE) [37] e a Programação Genética Estocástica baseada em Gramáticas (SG-GP) [33].

Este trabalho apresenta uma nova técnica que adapta a estimação de distribuição de probabilidade usada no BOA ao contexto da Programação Genética baseada em gramáticas livres de contexto.

De modo a estender o BOA para o contexto da PG, é usada uma representação linear de uma árvore de sintaxe concreta, semelhante à introduzida pela Evolução Gramatical [29]. A representação linear é facilmente obtida a partir da representação usada em programação genética, geralmente uma árvore de sintaxe abstrata. Esta conversão é necessária para atender os requerimentos do algoritmo de aprendizado de rede Bayesiana usado no BOA. A técnica foi aplicada em problemas tradicionais de programação genética para avaliar seu desempenho.

1.1 Computação Evolucionária

Computação Evolucionária (CE) é um ramo de pesquisa emergente da Inteligência Artificial, que propõe um novo paradigma para a solução de problemas inspirado na Seleção Natural [7].

A Computação Evolucionária compreende um conjunto de técnicas de busca e otimização. Para tanto, cria-se uma população de indivíduos que vão se reproduzir e competir entre si pela sobrevivência. Os melhores sobrevivem e transferem suas características às novas gerações. As técnicas atualmente incluem [4]: Programação Evolucionária, Estratégias Evolucionárias, Algoritmos Genéticos e Programação Genética. Estes métodos estão sendo utilizados, cada vez mais, pela comunidade de inteligência artificial para obter modelos de inteligência computacional [5]. A Computação Evolucionária pode ser utilizada para a obtenção de soluções para diversos problemas que, em geral, não possuem boas soluções com algoritmos comuns, devido à sua alta complexidade.

A Programação Genética (PG) tem como objetivo a geração automática de programas de computador para resolver um determinado problema. A partir de uma população inicial, geralmente criada de maneira aleatória, evolui-se uma população de soluções aplicando-se operadores genéticos, tais como cruzamento e mutação. O processo é guiado por uma função de aptidão (ou fitness) que mede o quanto o indivíduo é uma boa solução.

Indivíduos melhor adaptados têm maior chance de sobreviver. A PG será explicada mais detalhadamente no capítulo 2.

A Programação Genética vem sendo aplicada em diversas áreas do conhecimento, como Engenharia de Software, Circuitos Digitais, Mineração de Dados, Biologia Molecular e outras [12].

1.2 A Abordagem Probabilística

Em seu artigo [2] de 1994 intitulado "Population-Based Incremental Learning" (PBIL), Shumeet Baluja substituiu a população do algoritmo genético por um vetor de probabilidade que especificava as chances de cada posição de um vetor binário, codificante de uma solução, conter "1". O vetor de probabilidade pode ser visto como uma versão probabilística da população de indivíduos.

Em vez de lembrar a população, o vetor de probabilidade decide as áreas no espaço de busca onde a maioria dos novos indivíduos devem ser gerados. Essa abordagem pode ser considerada como uma forma diferente de representar uma grande população. Novas soluções são geradas usando a distribuição de probabilidade descrita no vetor. O algoritmo aprende ao atualizar o vetor de probabilidade segundo a conformação de um conjunto composto pelos melhores indivíduos da população. Em seus estudos [3] Baluja mostrou que PBIL tem performance superior ao algoritmo genético tradicional.

Apesar de o trabalho de Baluja se mostrar a fonte de referência quando se fala em evolução com representação probabilística da população, essa idéia já havia sido explorada no trabalho de Syswerda [38], onde ele propôs o Cruzamento Simulado Baseado em Bit (Bit-Based Simulated Crossover - BSC). Ambos BSC e PBIL são explemplos de algoritmos de distribuição marginal unidimencional (Univariate Marginal Distribution Algorithm - UMDA) [27], onde o aprendizado é feito somente se considerando parametros [24], desconsiderando possíveis relações entre os parâmetros.

A partir do PBIL surgiram uma grande gama de sistemas usando de alguma forma uma abordagem probabilística para representar a população. Entre esses trabalhos podemos citar BMDA [31], MIMIC [8], COMIT [1], FDA [28], cGA [16] e BOA.

Com o tempo, houve uma tendência dos novos algoritmos de explorar cada vez mais a inter-relação entre as variáveis aleatórias através de modelos probabilísticos cada vez mais poderosos e complexos, com o objetivo de obter melhores resultados. Esses trabalhos se concentram principalmente em aplicações tradicionais de Algoritmos Genéticos. No entanto, começam a surgir também técnicas que usam a representação por distribuição de probabilidade na área da Programação Genética, como PIPE [37] e SG-GP [33].

1.3 Esta Dissertação

1.3.1 Motivação

Nossa iniciativa foi motivada pelos bons resultados apresentados com diferentes técnicas que vêm surgindo e que usam modelos de distribuição probabilística. Somado a isso, havia o desejo de estudar se o algoritmo de evolução usando uma rede Bayesiana que foi apresentado no BOA conseguiria explorar a hierarquia implícita nas árvores que são usadas para representar um indivíduo na PG e se essa abordagem se mostraria melhor que a Programação Genética tradicional.

1.3.2 Objetivos

O objetivo é proporcionar um melhor entendimento sobre redes Bayesianas e EDAs, e propor uma técnica que combine o método de evolução e o modelo probabilístico propostos no BOA com a programação genética baseada em gramáticas, compondo uma ferramenta de programação automática que explore os benefícios de ambas as técnicas.

1.3.3 Estrutura do Trabalho

Esta dissertação está organizada da seguinte forma.

Capítulo 2 Uma introdução à Programação Genética, seus componentes básicos e uma introdução a Evolução Gramatical. Essas duas técnicas fornecerão conceitos básicos fundamentais que servirão de base para o desenvolvimento da técnica proposta.

Capítulo 3 Introduz o conceito de Algoritmos de Estimação de Distribuição e três técnicas que usam essa abordagem. Também é apresentado um resumo sobre redes Bayesianas e sobre o K2. Os conceitos apresentados nesse capítulo servirão de base para o desenvolvimento do modelo de aprendizado empregado na técnica proposta.

Capítulo 4 O algoritmo principal e todos os componentes básicos necessários para o composição da técnica proposta são apresentados.

Capítulo 5 Descrição e resultado dos testes realizados para avaliar o desempenho da técnica proposta.

Capítulo 6 São apresentadas as conclusões do tabalho e propostas de trabalhos futuros.

CAPÍTULO 2

PROGRAMAÇÃO GENÉTICA

Nesse capítulo serão apresentados conceitos básicos de programação genética, como seu algoritmo principal, representação de soluções usadas, particularidades e termos comuns da área que ajudarão no entendimento da técnica proposta.

2.1 Introdução à Programação Genética

A Programação Genética (PG) é um paradigma de inteligência artificial proposto por John Koza [21] com base nos trabalhos de John Holland [19] sobre Algoritmos Genéticos (AG). Atualmente ela representa uma importante área de estudo dentro de aprendizado de máquina devido a sua simplicidade e robustez. A Programação Genética tem sido aplicada nas mais diversas áreas onde métodos de busca por modelos complexos devam ser realizados em um espaço de busca.

A PG induz uma população de programas de computador, os quais melhoram sua capacidade de resolver um problema automaticamente conforme eles são expostos à dados mediante os quais eles são treinados [4]. Isso é, ela é um sistema de programação automática onde, dado um conjunto de símbolos com interpretação semântica, esses são combinados, a principio aleatoriamente e posteriormente usando critérios, para a busca de uma possível solução de um determinado problema.

Após a publicação do livro de Kosa, introduzindo e fundamentando a PG, surgiram muitos algoritmos que introduziram grandes modificações no algoritmo original [4]. No entanto, algumas características são comuns a todos esses sistemas, permitindo classificá-los como programação genética.

- Uso de processos estocásticos para a tomada de decisões em vários estágios da execução do problema.

- Construção de programas complexos a partir de unidades básicas chamadas terminais e funções. Essa construção acontece atualmente na inicialização do programa.

- Uso de operadores genéticos para modificar os programas construídos.

- Evolução de um conjunto de soluções propostas em paralelo usando para isso uma métrica de classificação de desempenho de cada solução proposta.

De forma geral, a Programação Genética funciona da seguinte maneira: um conjunto de possíveis soluções para um problema é gerado aleatóriamente. Essas possíveis soluções são avaliadas segundo um determinado critério e as características dos indivíduos promissores são misturadas para formar novos indivíduos. Esse processo se repete até que um critério de parada seja satisfeito. A figura 2.1 mostra uma versão esquemática do algoritmo geralmente empregado em PG.

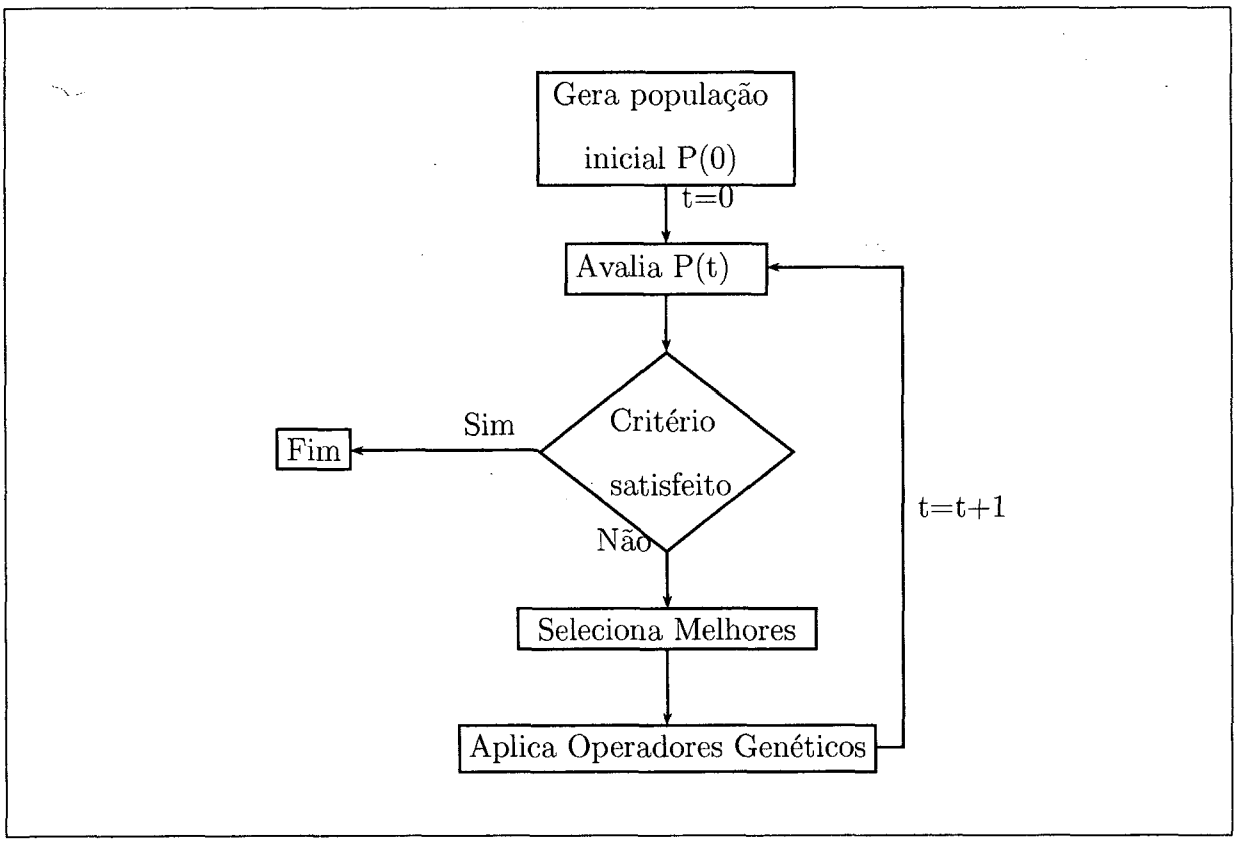


Figura 2.1: Algoritmo geral empregado na Programação Genética

2.2 Terminologia

A seguir, serão introduzidos termos comumente empregados em Programação Genética e em outros algoritmos evolucionários. Estes termos serão usados no decorrer desse trabalho.

A estrutura que representa uma possível solução para um problema é chamada indivíduo ou programa. Indivíduos que são representados através de vetores geralmente são chamados cromossomos. Cada elemento que compõe esse vetor é chamado gene.

Um conjunto de indivíduos forma uma população. Durante a execução do algoritmo, sucessivas populações são criadas. Para localizar uma determinada população no tempo se diz que uma população pertence a uma determinada geração.

Em PG, um indivíduo é composto por terminais e funções. Terminais englobam as variáveis e constantes de um programa. Funções são operadores que serão aplicados sobre os terminais. Como os indivíduos em PG geralmente são representados por árvores, ambos funções e terminais são referenciados como nós. O conceito de terminal em PG difere daquele usado comumente em gramáticas. Nessas, terminais compreendem variáveis, constantes e operadores, como será explicado na seção 2.7.2.

Uma população evolui através do emprego de operadores genéticos. Operadores genéticos são procedimentos que modificam um ou mais indivíduos segundo um critério. Dois operadores genéticos amplamente utilizados são o cruzamento e a mutação. Esses processos serão melhor explicados na seção 2.6

De forma a diferenciar os indivíduos com relação à sua eficiência em revolver um determindo problema é definida uma métrica. À métrica definida é dada o nome de função de aptidão.

2.3 Representação do Problema

A representação de um problema define o conjunto de todas possíveis soluções para um problema que um sistema em particular pode encontrar [4]. Essa representação pode ser dividida em três componentes: um sintático, um semântico e um computacional. O com-

ponente sintático está associado com convenções notacionais de uma dada representação. O componente semântico está relacionado com as convenções de interpretação de uma representação. Por fim o componente computacional é responsável por especificar como os objetos e relações são manipulados de acordo com as convenções semânticas [9].

Na PG, o componente sintático é formado pelos terminais, funções e uma estrutura de organização que permite que ambos os elementos sejam aplicados nos momentos adequados e de forma apropriada, assim como deve permitir o uso de operadores genéticos.

A estrutura comumente usada em PG para organizar a representação de um problema é uma árvore de sintaxe abstrata, que permite que os indivíduos sejam formados pela livre combinação de funções e terminais. Nesse modelo de representação uma aridade deve ser atribuída a cada função.

Duas características importantes quando se modela um problema na PG são a suficiência e o fechamento.

O fechamento postula que as funções devem conseguir tratar todos os possíveis parâmetros de entrada. Isso quer dizer que em um modelo onde estivesse sendo usada a operação de divisão, a função responsável pela divisão deve ser capaz de tratar divisões por zero sem gerar erros. Isso também permite que duas sub-árvores de diferentes árvores, cujos nós raizes das sub-árvores sejam funções, sejam permutadas [40].

A suficiência requer que funções e terminais devam ter poder de descrição suficiente para modelar o problema, caso contrário o sistema não será capaz de encontrar uma solução [21].

Caso se desejasse criar uma descrição de polinômios usando uma árvore, teria-se funções de aridade dois relativas à soma e à multiplicação (+, *) e terminais representando variáveis e constantes, (x, 2).

Para efeito de melhor visualização, é comumente empregada uma representação gráfica da árvore, onde nós são conectados por arestas. Por exemplo, o polinômio x*x+2*x+2 tem como uma de suas representações gráficas a figura 2.2.

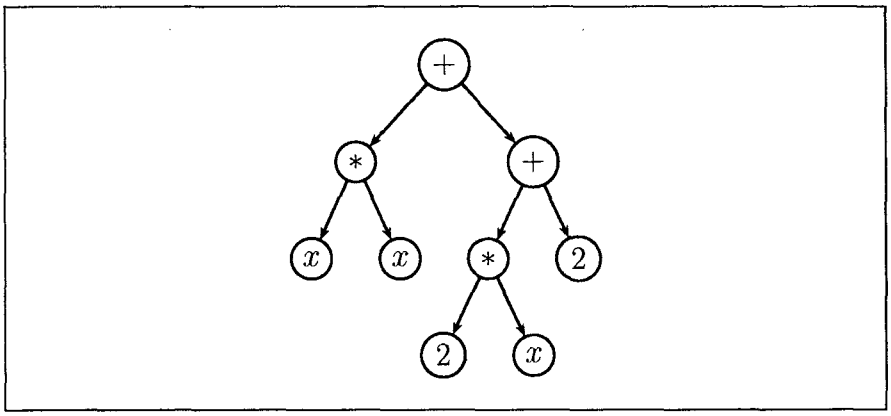


Figura 2.2: Árvore de sintaxe abstrata de x*x+2*x+2

O componente semântico em programação genética é representado por um processo de avaliação, que será explicado na seção 2.4. O componente computacional é representado pelos operadores genéticos, que serão explicados na seção 2.6

2.4 Avaliação

Segundo a teoria da evolução de Darwin, indivíduos que se adaptam melhor ao seu ambiente têm maior chance de passar suas características para gerações seguintes. Essa prerrogativa é usada pela Programação Genética para promover a convergência dos indivíduos da população para uma solução ótima. Para tanto, é necessária a definição de uma métrica que classifique cada indivíduo segundo a sua aptidão para resolver o problema proposto.

Essa métrica é definida através de uma função de aptidão. A função de aptidão é considerada um fator crítico no sucesso de um sistema de PG. Sua principal característica é ser capaz de diferenciar quais os melhores indivíduos de uma população.

A necessidade de uma função de aptidão é um dos fatores que limitam a criação de sistemas genéricos de programação genética, já que ela é especifica do domínio do problema.

2.5 Seleção

A seleção é o processo responsável por escolher a quais indivíduos de uma população serão aplicados os operadores genéticos. Ela controla a velocidade da evolução e é freqüentemente apontada como a responsável quando um algoritmo evolucionário converge prematuramente [4].

A escolha é feita usando-se como parâmetro o valor retornado pela função de aptidão que determinou a qualidade de cada indivíduo. Existe uma grande variedade de métodos de seleção, cada qual favorecendo uma determinada característica da população.

Alguns desses métodos são:

- Elitísmo - Esse método seleciona os indivíduos com o melhor valor de aptidão da população.

- Torneio - Esse método seleciona aleatóriamente um conjunto de indivíduos dentro da população e seleciona entre esses o de maior aptidão.

2.6 Operadores Genéticos

Os operadores genéticos são responsáveis por alterar os indivíduos selecionados, promovendo a sua evolução e compondo uma nova população. Os operadores são aplicados em um conjunto de um ou dois indivíduos da população, dependendo do operador, selecionados através de um dos métodos de seleção como os listados na seção 2.5. Os métodos mais comumente empregados para esse propósito são: a mutação, o cruzamento e a reprodução.

- Cruzamento - O cruzamento troca componentes de dois indivíduos, formando dois novos indivíduos. O cruzamento está representado na figura 2.3.

- Mutação - A mutação remove um dos componentes do indivíduo e o substitui por um outro criado aleatoriamente. Um exemplo de mutação está representado na figura 2.4.

- Reprodução - A reprodução preserva um indivíduo integralmente para a próxima geração.

Os operadores genéticos devem ser construídos de forma a manter a validade sintática dos cromossomos modificados.

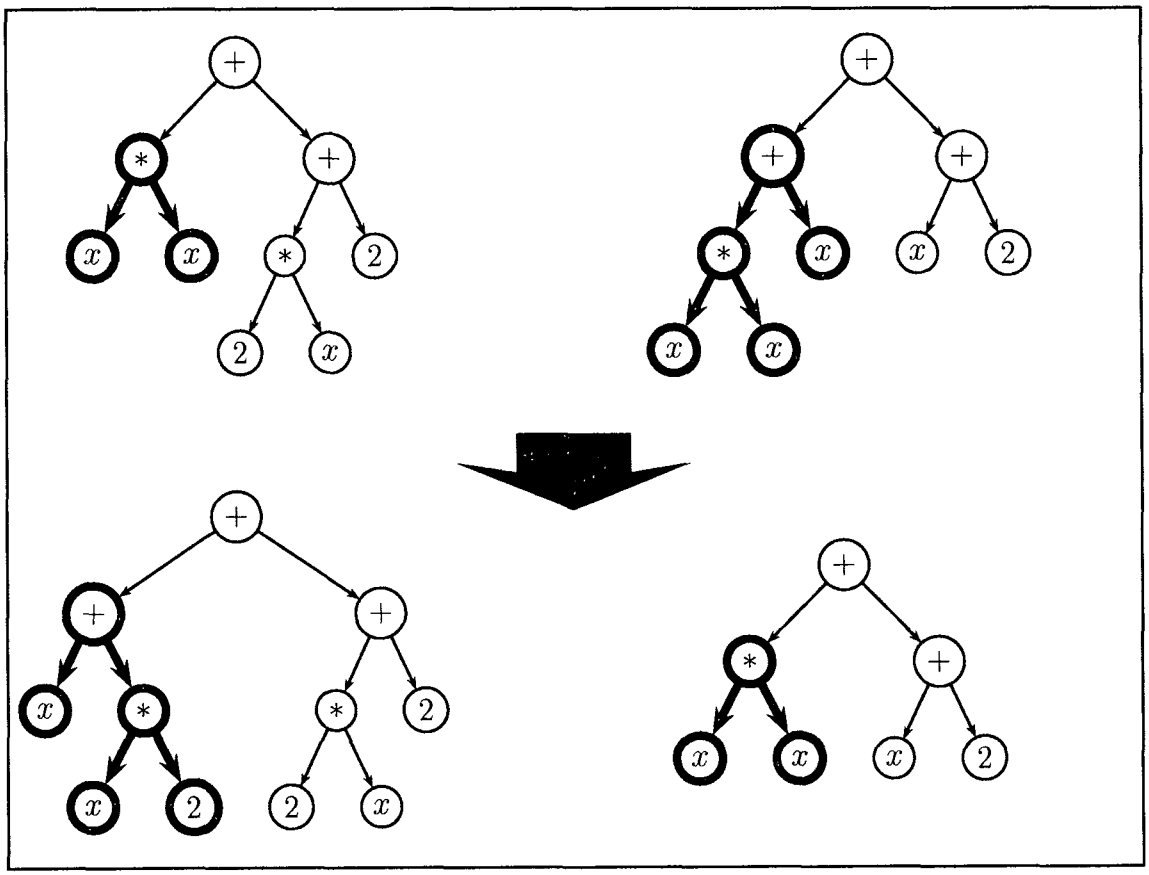


Figura 2.3: Duas árvores tem um de seus componentes trocados pela operação de cruzamento

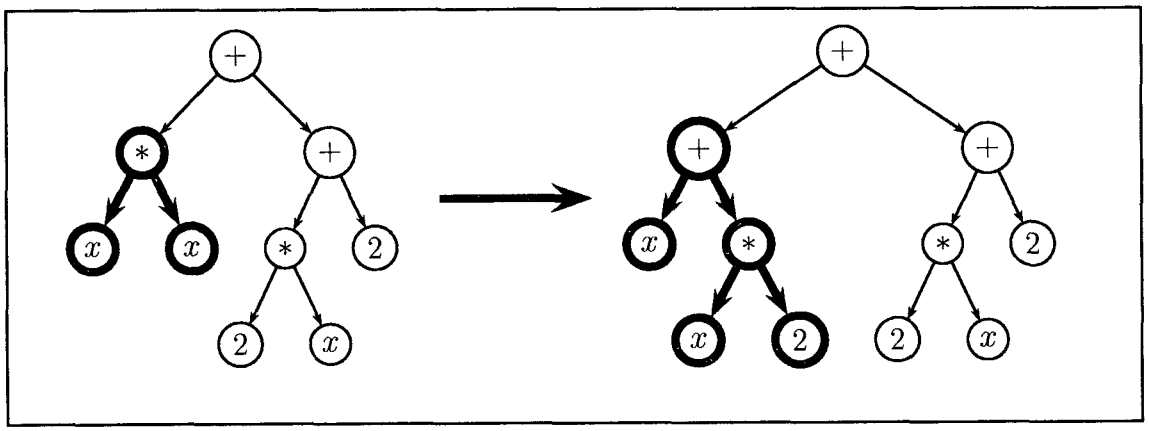


Figura 2.4: Exemplo do processo de mutação

2.7 Programação Genética baseada em GLC

2.7.1 Gramáticas

Uma gramática é uma 4-tupla (N, T, P, S) tal que N e T são conjuntos finitos de símbolos, $N \cap T = \oslash$, P é um conjunto de pares (R, Q) tal que $P \in N$ e $Q \in (N \cup T)^*$, e $S \in N$ [40].

Gramáticas podem ser usadas para gerar sentenças de uma linguagem. Para gerar uma sentença derivada de G é necessário iniciar com S, escolher uma regra de formação $S \rightarrow \alpha$ e sucessivamente ir expandindo o não-terminal em α até que restem apenas terminais. A figura 2.5 contém um exemplo de gramática.

$$\text{N}=\{\text{K,G}\},\ \text{T}=\{\sigma,\omega\}\ \text{e}\ \text{S}=\{\text{K}\}$$

$$\text{P} = \begin{cases} \text{K} \rightarrow \text{G}\sigma \\ \text{G} \rightarrow \omega\omega \end{cases}$$

Figura 2.5: Um exemplo de uma gramática.

As formas intermediárias do processo de produção, onde existem não-terminais e terminais, são chamadas formas sentenciais. Se uma forma sentencial possui apenas terminais ela é chamada sentença. As transições de uma forma sentencial para uma outra, através da substituição de um de seus não-terminais, são chamadas passos de produção e as regras são freqüentemente chamadas regras de produção.

O processo de produção pode ser representado graficamente desenhando-se linhas entre símbolos correspondentes. Tais gráficos são chamados grafos de produção porque eles descrevem os passos que levam a composição da sentença final.

2.7.2 Gramáticas Livre de Contexto

Uma gramática dita livre de contexto (GLC) contém apenas um não-terminal no lado esquerdo de suas regras de produção. Como há apenas um símbolo no lado esquerdo das regras de produção, nós vizinhos são independentes com relação a suas possíveis produções. Dessa forma, qualquer nó do grafo possui no máximo um nó pai, o que conseqüentemente faz com que o seu grafo de produção seja uma árvore.

Um exemplo de uma gramática livre de contexto é ilustrada pela figura 2.6.

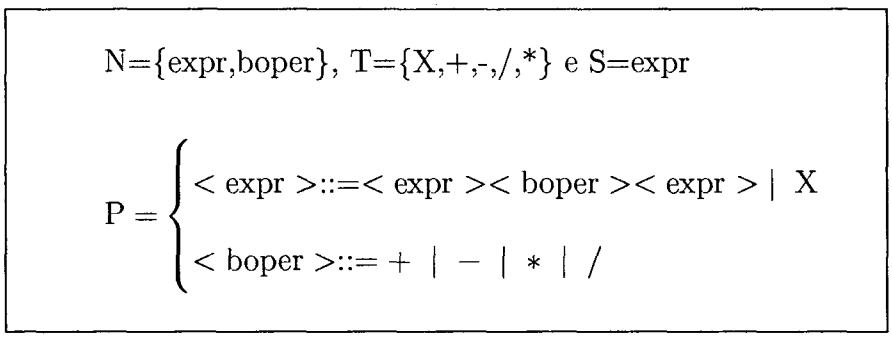


Figura 2.6: Um exemplo de uma gramática livre de contexto em notação BNF.

A representação em árvore de uma sentença gerada de uma GLC é chamada árvore de sintaxe abstrata. A figura 2.7 mostra todos os passos para a construção de uma sentença derivada da gramática da figura 2.6 usando redução mais a esquerda. Note que nesse tipo de redução é sempre reduzido o símbolo não-terminal que aparece mais a esquerda na expressão.

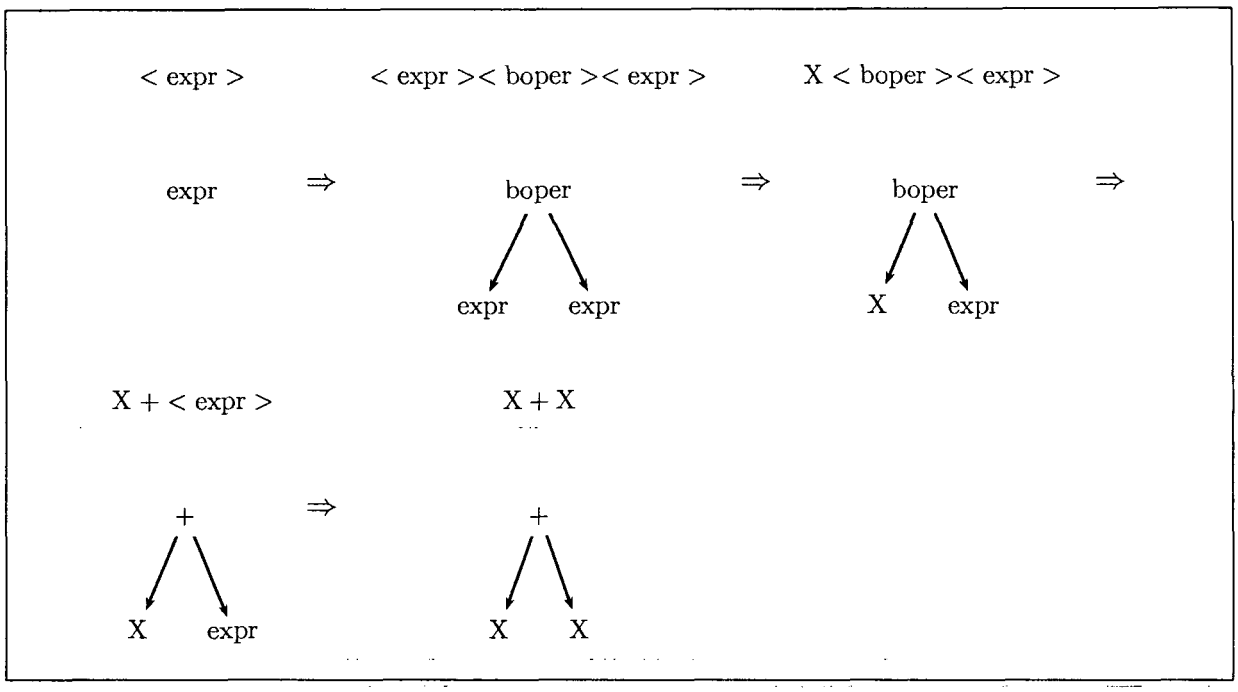


Figura 2.7: Os passos de produção usados na construção de uma expressão simples $(x+x)$ usando uma GLC.

2.7.3 Características da PG baseada em GLC

Muitos programas requerem funções com argumentos tipados. Nesses casos é necessário que restrições sejam aplicadas no processo de criação da população inicial e aos operadores genéticos para garantir a satisfatibilidade da propriedade do fechamento, enunciando na seção 2.3. Para atender às restrições sintáticas e ainda assim se manter como uma ferramenta genérica, Frederic Gruau [11] propôs o uso de gramáticas, originando o que veio a se tornar a programação genética baseada em gramáticas livres de contexto.

Na programação genética baseada em GLC, uma gramática livre de contexto é usada para guiar a criação dos indivíduos da população inicial. Esses indivíduos estão representados por uma árvore de sintaxe concreta ao invés de uma árvore de sintaxe abstrata. Essa representação permite a fácil manutenção da sintaxe da árvore que vá sofrer alguma operação genética dado que os seus nós estão classificados segundo as definições da gramática. Assim as operações que substituem um componente cuja raiz é de um determinado não-terminal mantêm a consistência da árvore ao substituir esse nó por um

componente cuja raiz seja dessa mesma classe gramatical.

O uso de GLC permite a expressão de restrições especificas de um problema no espaço de busca da PG [33].

Figura 2.8 exemplifica uma árvore de sintaxe concreta e sua árvore de sintaxe abstrata geradas segundo a gramática da figura 2.6.

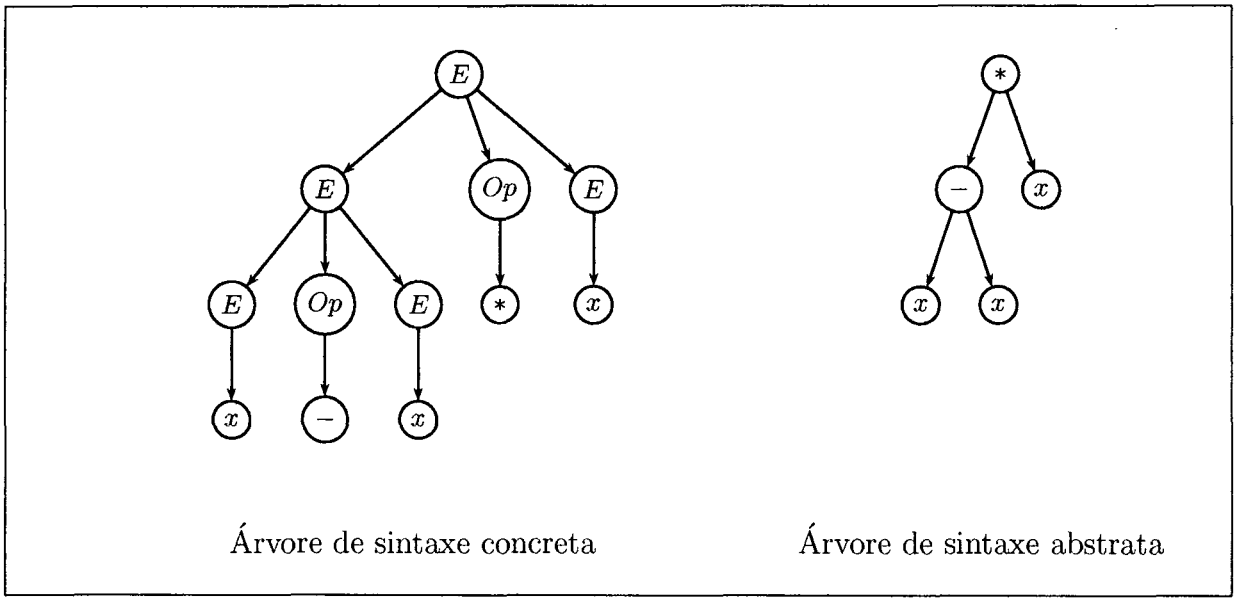


Figura 2.8: A árvore de sintaxe concreta e a árvore de sintaxe abstrata da expressão $(x - x) * x$.

2.8 Evolução Gramatical

A Evolução Gramatical (Gramatical Evolution - GE) é uma técnica de Algoritmos Evolucionários que pode evoluir programas codificados em uma linguagem qualquer através de um vetor de números binários de tamanho fixo [29] .

A Evolução Gramatical tenta aproximar a programação genética do modelo biológico ao separar a representação do genótipo do fenótipo. Isso é obtido através do uso de um vetor binário de tamanho fixo, chamado cromossomo. Esse vetor é o equivalente na célula ao DNA, ou seja, o genótipo. Assim como o DNA, ele deve ser adequadamente decodificado para que gere um programa, o fenótipo, que efetivamente possa resolver um determinado problema.

A geração de um programa a partir de um vetor de números binários é feita usando uma gramática livre de contexto cujas regras de formação estão enumeradas. Para cada não-terminal, todas as suas possíveis produções são enumeradas seqüencialmente iniciando de zero. A figura 2.9 mostra uma gramática enumerada dessa forma.

N={ expr,boper,uoper} , T={X, +, -, %, *, sen, cos, log, exp}, S={expr}

$P =$

					Enumeração
< expr >	::=	< expr >	< boper >	< expr >	(0)
	|	< uoper >	< expr >		(1)
	|	X			(2)
< boper >	::=	+			(0)
	|	-			(1)
	|	*			(2)
	|	/			(3)
< uoper >	::=	sen			(0)
	|	cos			(1)
	|	log			(2)
	|	exp			(3)

Figura 2.9: Gramática cujas regras de formação foram enumeradas.

O cromossomo tem seus números binários agrupados em unidades de tamanho fixo previamente determinado pelo usuário. Esses agrupamentos são convertidos para números inteiros vindo a compor um vetor de inteiros. A figura 2.10 mostra a conversão de um vetor de números binários para um vetor de números inteiros como proposto na Evolução Gramatical.

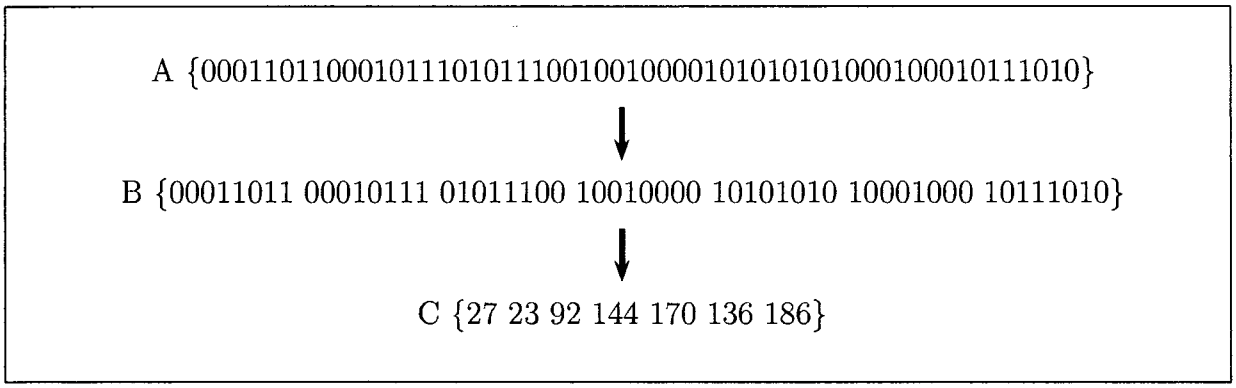


Figura 2.10: Conversão de um vetor de números binários (A) em um vetor de números inteiros (C). Os números inteiros foram produzidos pela conversão do número formado pelo agrupamento de oito números binários para a base dez.

Um programa é criado usando o processo de produção de uma sentença como já introduzido na seção 2.7.2, com a diferença de que o vetor de inteiros é usado para decidir qual produção será selecionada sempre que um não-terminal precisar ser traduzido.

Toda vez que a tradução de um não-terminal é necessária, o vetor é consultado. Para cada consulta, um inteiro distinto x é selecionado seqüencialmente do início do vetor para o fim, e passado como parâmetro à equação 2.1. A regra da gramática cujo número é igual ao valor retornado pela função substuirá o não-terminal.

$$f(x) = x \; modulo \; N \tag{2.1}$$

Onde N é o número de possíveis produções do não-terminal na gramática.

Dado a gramática G da figura 2.9, um cromossomo já convertido para um vetor de inteiros C e um vetor auxiliar I, que pode conter terminais e não-terminais, a produção ocorre como segue:

Ao final desses passos I poderá conter um programa válido. Um programa é dito válido se ele é composto somente de terminais.

Após o processo de decodificação do cromossomo ele é avaliado pela função de aptidão. Os cromossomos que são decodificados em programas não válidos recebem o pior valor de aptidão possível retornado pela função de aptidão.

Algoritmo 1 Decodificação de um cromossomo.

1: O processo tem início com a concatenação de S em I, inicialmente vazio.

2: i recebe 1.

3: O vetor I é percorrido da esquerda para a direita em busca de um não-terminal.

4: Se um não-terminal é encontrado, a gramática é usada para traduzi-lo em uma de suas produções segundo a regra retornada pela função 2.1, que recebe como parâmetro $c_i \in C$.

5: O não-terminal em I é substituído pelo lado direito da regra de produção.

6: $i = i + 1$.

7: Se I possui não-terminais e $i \leq \mid C \mid$, volte ao passo 3.

Fora o sistema de codificação do cromossomo e sua posterior decodificação em um programa, a GE segue o mesmo algoritmo geral da programação genética introduzido na seção 2.1, com modificações nos operadores genéticos para se adequarem à representação por vetor de inteiros.

A figura 2.11 mostra a comparação entre o modelo biológico e o modelo proposto em Evolução Gramatical [29].

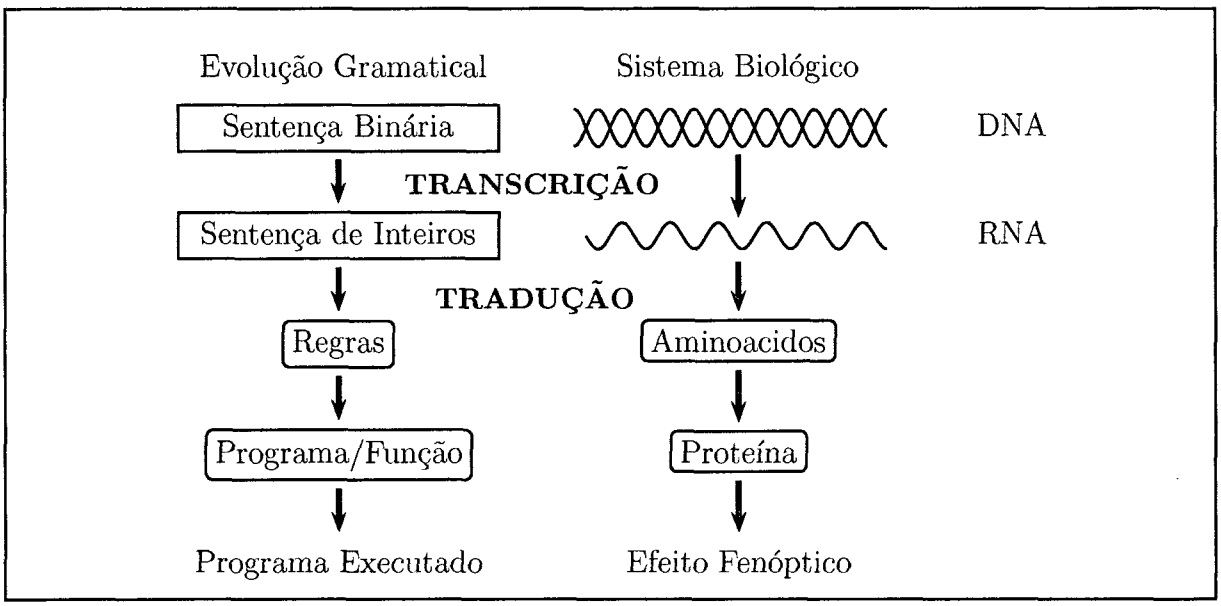


Figura 2.11: Analogia entre o modelo biológico e o de Evolução Gramatical.

É importante notar um problema no método proposto na GE, e que influenciará posteriormente a representação e produção dos cromossomos da técnica que será proposta.

O problema é a distribuição probabilistica irregular entre as regras de produção que traduzem um não-terminal. Ele surge quando uma gramática possui mais que um não-terminal e cada não-terminal possui várias regras derivando dele. Nesse caso, o tamanho do agrupamento de números binários que compõem o cromossomo e que formarão o vetor de inteiros, sofre algumas limitações. Note que os inteiros formados vão de zero até uma potência de dois menos um.

Para se obter uma distribuição probabilística uniforme entre todas as regras de formação, o maior inteiro resultante da conversão do agrupamento de números binários deve ser um múltiplo comum do número de regras derivadas de cada não-terminal. Caso contrário, algumas regras terão probabilidades maiores que as outras. Tomemos como exemplo a gramática descrita na figura 2.9.

A gramática possui três não-terminais. O primeiro possui três regras que derivam dele e os outros quatro. Caso um número diferente de um dos múltiplos de doze seja escolhido como inteiro máximo para cada gene, o que obrigatoriamente ocorrerá, dado que doze não é uma potência de dois, a distribuição de probabilidade entre as regras de formação entre cada não-terminal não será uniforme. Por exemplo, se escolhermos o número sete como valor máximo para cada gene e desejassemos escolher uma das regras de formação que derivam de $< expr >$, a probabilidade de escolher a regra 0 ou 1 é de 38%, no entanto, a de escolher a regra 2 é de 25%. Essa diferença de probabilidade na escolha das produções faz com que alguns indivíduos sejam gerados mais que outros.

Esse efeito poderia ser evitado modificando-se a gramática, para que os não-terminais tenham um número conveniente de produções, ou escolhendo um número máximo para um gene que fosse grande o suficiente para reduzir a diferença de probabilidade entre as regras. A primeira opção não é adequada, pois a gramática ficaria limitada e poderia ter sua capacidade de introduzir conhecimento a priori comprometida. Logo, a melhor opção seria a escolha de números grandes.

Nesse capítulo foram apresentados conceitos básicos de Programação Genética, uma introdução sobre GLC, e como ela pode ser aplicada para estender a capacidades da PG. Também introduzimos a Evolução Gramatical e algumas de suas limitações. A seguir

serão apresentados conceitos relativos a Algoritmos de Estimação de Distribuição, que em conjunto com conceitos da PG e da GE formarão a base para a composição da técnica a ser proposta.

CAPÍTULO 3

ALGORITMOS DE ESTIMAÇÃO DE DISTRIBUIÇÃO

Nesse capítulo introduziremos os conceitos básicos de Algoritmos de Estimação de Distribuição (Estimation of Distribution Algorithm - EDA), alguns exemplos de ferramentas que usam essa abordagem e um modelo probabilístico.

Algoritmos de Estimação de Distribuição são algoritmos de busca baseados em populações que usam modelamento probabilístico de soluções promissoras em combinação com a simulação de modelos induzidos para guiar suas buscas [25].

Esses algoritmos tentam superar algumas limitações encontradas em AG e que também ocorrem na PG. Uma dessas limitações são os parâmetros que o usuário deve definir, como taxa de mutação, taxa de cruzamento, tamanho da população, tipos de operadores, etc, e que geram certo grau de dificuldade para usuários que não têm conhecimento adequado sobre essas técnicas [25]. A segunda é o problema de acoplamento [30], que ocorre porque essas técnicas têm dificuldade em identificar e preservar componentes de um indivíduo que não devem ser quebrados em partes.

A primeira limitação é resolvida apenas parcialmente. Os EDAs geralmente não possuem operadores genéticos, já que eles usam uma distribuição de probabilidade para gerar novos indivíduos. No entanto, a forma como os indivíduos serão representados, o tamanho da população e a definição de uma função de aptidão adequada são tarefas que ainda dependem do usuário.

A segunda limitação é melhor superada pelos EDAs, através do uso de uma distribuição de probabilídades, para descrever as inter-relações entre indivíduos selecionados de uma população. A distribuição permite conservar componentes que agregam alto valor aos indivíduos dos quais eles fazem parte.

Assim como os algoritmos evolucionários, os EDAs são algoritmos que empregam uma busca em paralelo através de uma população de possíveis soluções. Mas ao contrário das

técnicas de Computação Evolucionária, a evolução não é promovida através da aplicação de operadores genéticos. Ao invés, esses algoritmos usam uma distribuição de probabilidade estimada a partir de um conjunto de dados composto por indivíduos da população anterior.

Enquanto que em Computação Evolucionária as inter-relações entre as diferentes variáveis que compõem os indivíduos são implicitas, em EDAs as inter-relações são representadas de forma explícita através da distribuição de probabilidade conjunta associada com os indivíduos selecionados em cada interação [25]. O custo computacional da estimação da distribuição de probabilidade conjunta a partir da base de dados contendo os indivíduos selecionados constitui o fator limitante dessa nova heurística. O pseudocódigo abaixo mostra o algoritmo geral de um típico EDA [41].

Algoritmo 2 Algoritmo geral de um EDA.

1: Gere a população inicial aleatóriamente.
2: Selecione alguns indivíduos da população atual de acordo com o método de seleção.
3: Construa o modelo de probabilidade usando os indivíduos selecionados.
4: Substitua alguns ou todos os membros de população atual por novas soluções derivadas do modelo de probabilidade.
5: Se a condição de parada não for satisfeita, vá para o passo 2.

Os EDAs podem ser divididos em três diferentes abordagens [23].

Variáveis Independentes: Os Algoritmo de Distribuição Marginal Unidimencional (UMDA), Aprendizado Incremental Baseado em População (Population-Based Incremental Learning - PBIL) e o Algoritmo Genético Compacto (Compact Genetic Algorithm - cGA) estão nesse grupo. Esses algoritmos calculam a estimação da distribuição de probabilidade considerando as variáveis do problema como unidimencionais. Eles apresentam melhor desempenho em problemas onde as variáveis não possuem interações significativas entre elas.

Dependências Bidimencionais: Os algoritmos Agrupamento de Entradas por Maximização de Informações Mutua (Mutual Information Maximization for Input Clustering - MIMIC), Otimizadores Combinantes com Árvores de Informação Mutua (Combining Op-

timizers with Mutual Information Trees - COMIT) e Algoritmo de Distribuição Marginal Bidimencional (Bivariate Marginal Distribution Algorithm - BMDA) pertencem a esse grupo. Esses algoritmos exploram as dependências entre no máximo duas variáveis para prover melhores resultados em problemas cujas variáveis estão relacionadas.

Dependências Multidimencionais: Os algoritmos Algoritmo de Distribuição Fatorada (Factorized Distribution Algorithm - FDA), Algoritmo Genético Compacto Extendido (Extended compact Genetic Algorithm - EcGA) [14], Algoritmo de Otimização Bayesiano (Bayesian Optimization Algorithm - BOA) e Algoritmo de Estimação de Rede Bayesiana (Estimation of Bayesian Network Algorithm - EBNA) [23] estão nesse grupo. Esses algoritmos são capazes de capturar dependências entre quaisquer número de variáves.

A seguir serão introduzidos dois algoritmos que usam distribuição de probabilidade para a programação automática, o PIPE e a SG-GP e um algoritmo de otimização, BOA, que foi usado como base para o desenvolvimento da nossa ferramenta. As introduções aqui apresentadas não têm como objetivo discutir em detalhes nenhum dos algoritmos, mas apenas fornecer uma visão geral focalizada em itens que suportem o trabalho aqui apresentado.

3.1 Evolução Incremental Probabilística de Programas

A Evolução Incremental Probabilistica de Programas [37] (Probabilistic Incremental Program Evolution - PIPE) é uma técnica para a síntese automática de programas, assim como a Programação Genética. O método de aprendizado aplicado em PIPE é inspirado em PBIL, já introduzido na seção 1.2, o qual foi estendido e modificado para se adequar ao contexto da programação automática.

PIPE propõe o uso de uma estrutura de dados chamada *árvore protótipo probabilística* (probabilistic prototype tree - PPT), cuja estrutura é semelhante à representação por árvore da Programação Genética. Mas ao contrário da árvore de PG, cada nó de uma PPT possui uma tabela descrevendo a probabilidade de um conjunto de instruções. O PIPE aprende a solução de um problema modificando a PPT de forma a incrementar a probabilidade de produzir o melhor indivíduo encontrado em cada geração. Para se adap-

tar a estrutura do melhor indivíduo, a PPT pode crescer ou encolher. As probabilidades das instruções podem ainda ser modificadas aleatóriamente para melhor explorar o espaço de busca.

3.2 Programação Genética Estocástica baseada em Gramáticas

A Programação Genética Estocástica baseada em Gramáticas (Stochastic Grammar-based Genetic Programming SG-GP) [33] é uma ferramenta de programação automática que usa um modelo de distribuição de probabilidade para tentar evitar o crescimento excessivo de código inútil, como acontece na Programação Genética. Esse crescimento se dá devido à incorporação de trechos não codificantes no código do indivíduo, conhecidos como **introns**.

A SG-GP também emprega uma gramática livre de contexto para introduzir restrições especificas do problema na formação dos cromossomos de forma a restringir o espaço de busca [32].

O modelo de distribuição de probabilidade é representada por um vetor que contém todas as possíveis formações cuja raiz seja um não-terminal e cujo tamanho seja menor que um valor máximo. Essas produções são construídas segundo a gramática usada. A cada formação é atribuído um peso que descreve a probabilidade dele ser selecionado. O algoritmo aprende atualisando esses pesos segundo a sua freqüência num conjunto de indivíduos selecionados da população atual.

Um novo indivíduo é criado selecionando produções segundo o modelo de distribuição e substituindo os não-terminais nas folhas da árvore por outras produções quando necessário. Em testes, SG-GP foi capaz de evitar a formação de **introns** [33]. Assim como PBIL, SG-GP usa apenas infomação unidimencional em seu modelo de distribuição.

3.3 Rede Bayesiana

Uma rede Bayesiana é um modelo para relacionamento probabilístico sobre um conjunto de variáveis [18]. Ele é composto por um vetor de variáveis aleatórias $\mathbf{X} =$

$(X_1, ..., X_n)$, que podem ser discretas ou contínuas, e um grafo acíclico $G = (\mathbf{X}, E)$ cujas arestas $E \subseteq [\mathbf{X}]^2$ representam uma relação de dependência.

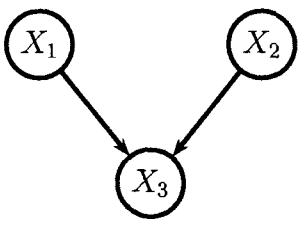
Variáveis aleatórias são funções de $\omega \in \Omega$ para um domínio representando as qualidades ou distinções de interesse, onde Ω é um espaço amostral [22]. Se há uma aresta saindo da variável aleatória X_i e chegando em X_j, X_i é dito pai de X_j. A probabilidade condicional de X_j dado X_i é representa por $p(X_j \mid X_i)$

Dado um grafo G, a distribuição de probabilidade conjunta para o vetor $\mathbf{X}$ é dada por

$$p(x) = \prod_{i=1}^{n} p(x_i \mid pai(X_i)) \tag{3.1}$$

Uma rede Bayesiana modela eficientemente a distribuição de probabilidade conjunta de um grande conjunto de variáveis através do par $(\mathbf{X}, G)$. Enquanto que G descreve a topologia da rede, $\mathbf{X}$ contem os parâmetros que compõem as tabelas de probabilidade condicional (TPC).

Uma tabela de probabilidade condicional armazena a chance que um evento irá acontecer dado que um outro evento já aconteceu. Cada linha em uma TPC contém a probabilidade condicional de cada valor no nó para todas as combinações de valores nos nós que são seus pais [36]. A figura 3.1 mostra as TPCs de uma rede Bayesiana com três variáveis aleatórias X_1, X_2 e X_3, onde X_1 e X_2 são pais de X_3 e cada variável aleatória possui dois valores possíveis, 0 e 1.



X_1

1	0
0,2	0,8

X_2

1	0
0,6	0,4

X_3

		X_3	
X_1	X_2	1	2
1	1	1,0	0,0
1	0	0,7	0,3
0	1	0,1	0,9
0	0	0,5	0,5

Figura 3.1: Exemplo das tabelas de probabilidade condicional de três variáveis aleatórias.

Usando a descrição do domínio através de uma rede Bayesiana, é possível gerar novas instâncias das variáveis com propriedades similares àquelas dos dados de treinamento

[30].

3.3.1 Aprendendo a Rede Bayesiana a Partir de Dados

Tanto a topologia quanto os parâmetros de uma rede bayesiana podem ser aprendidos através de dados [17] automaticamente.

Há dois componentes dos algoritmos de aprendizado de estrutura: uma métrica de pontuação e um procedimento de busca [17]. Uma métrica de pontuação é uma medida de quão bem a rede modela os dados. Conhecimento a priori pode ser incorporado à métrica para reduzir o seu tempo computacional.

O procedimento de busca explora o espaço de todas possíveis redes de modo a encontrar a rede (ou um conjunto de redes) que maximiza o valor da métrica de pontuação. O número de redes candidatas a melhor rede pode ser reduzido por operadores de restrição [30].

Na Programação Automática Bayesiana, nós aplicamos o algoritmo heurístico K2 [6] para buscar a melhor entre todas as possíveis estruturas. O K2 usa uma abordagem por modelo de seleção, onde uma métrica de pontuação guia um algoritmo guloso simples na busca pela melhor rede. Baseado em quatro suposições, o K2 usa uma formula para computar a probabilidade de uma rede contendo apenas as variáveis de uma base de dados de registros. As quatro suposições são:

1. As variáveis na base de dados são discretas.

2. Casos ocorrem independentemente, dado um modelo de rede de Bayes.

3. Não são omitidos valores na base de dados.

4. A função de densidade $f(B_p \mid B_s)$ é uniforme. B_p é um vetor cujos valores denotam uma atribuição de probabilidade condicional associada com a estrutura B_s.

Para trazer a complexidade computacional para tempo polinomial, mais três suposições são assumidas.

5. Os nós estão ordenados de forma a que nenhum nó pode ser pai de um nó que o precede em ordem. Ou seja, o grafo não contém ciclos.

6. Há um limite reduzido para o número máximo de pais (k) que cada nó pode ter.

7. Dado que π é pai de x, $p(\pi_i \mid x_i)$ e $p(\pi_j \mid x_j)$ são independentes quando $i \neq j$.

Usando essas sete suposições o K2 procura por uma estrutura de rede Bayesiana B_s que maximize a equação abaixo, dada uma base de dados D.

$$max[P(B_s, D)] = \prod_{i=1}^{n} P(\pi_i \rightarrow x_i) \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} N_{ij} \prod_{k=1}^{r_i} N_{ijk}! \tag{3.2}$$

Onde, r_i são todos possíveis eventos de uma variável x_i, π_i é um conjunto de variáveis que são pais de x_i, q_i é um conjunto de únicas instanciações de π_i, w_i denota a j-ésima instanciação de π_i relativa a D, N_{ijk} é o número de casos em D onde as variáveis x_i tem o valor de v_{ik} e π_i está instanciado com w_{ij}.

A estrutura é construída adicionando arestas a uma rede sem arestas. Conexões que maximizam a métrica de pontuação são adicionados ao grafo. Somente operações que mantêm a rede acíclica e com no máximo k arestas de entrada em cada nó são permitidas [30]. K2 tem ordem de complexidade para o aprendizado da estrutura da rede de $O(m\ u^2\ n^2\ r)$, onde m é o número de casos, u é o número máximo de pais que cada nó pode ter, n é o número de nós e r é o número de possíveis eventos [6].

Além de fornecer a estrutura da rede, K2 prove uma tabela de probabilidade condicional para cada $X_i \in \mathbf{X}$. As TPCs fornecem as probabilidades que serão usadas pelo algoritmo de geração de cromossomos para a construção de uma nova população.

3.4 Algoritmo de Otimização Bayesiana

O Algotitmo de Otimização Bayesina (BOA) é um algoritmo de otimização baseado nos conceitos de Algoritmos Genéticos. Ele usa uma distribuição de probabilidade estimada de soluções promissoras para gerar novas soluções. Para estimar a distribuição, são usadas técnicas para modelar dados multidimencionais por redes Bayesianas [30].

Assim como um Algoritmo Genético comum, o BOA usa uma população de cromossomos representados por vetores binários para modelar um problema. Após a criação de uma população inicial gerada aleatoriamente e sua avaliação, os melhores indivíduos são selecionados e deles é estimada uma distribuição de probabilidade. Essa distribuição é modelada através de uma rede Bayesiana cuja estrutura e probabilidades são aprendidas usando o algoritmo K2. O algoritmo 3 mostra o pseudocódigo do BOA.

Algoritmo 3 Algoritmo geral do BOA

1: t recebe 0.
2: Aleatoriamente gere uma população inicial $P(0)$.
3: Selecione um conjunto de vetores promissores $S(t)$ de $P(t)$.
4: Construa a rede B usando uma métrica escolhida e restrições.
5: Gere um conjunto de novos vetores $O(t)$ de acordo com a distribuição conjunta codificada em B.
6: Crie uma nova população $P(t+1)$ substituindo alguns vetores de $P(t)$ com $O(t)$.
7: t recebe $t+1$.
8: Se o critério de parada não for satisfeito, vá para o passo 3.

Ao usar uma rede Bayesiana para modelar a distribuição probabilística da população, o algoritmo explora a capacidade desse modelo em correlacionar variáveis que possuem relações de condição. Apesar de usar o algoritmo K2 para o aprendizado da estrutura da rede, o BOA não usa adequadamente a informação de ordenação das variáveis aleatórias. Ao invés, ele dispõe as variáveis dispostas aleatoriamente. O ordenação das variáveis aleatórias usando um critério se mostra um fator importante quando se usa o K2, como mostrado em [20].

Nos testes realizados, o BOA se mostrou superior ao algoritmo genético tradicional em problemas de otimização e foi capaz de evitar o problema de acoplamento [30]. Esses resultados nos motivaram a tentar estender o mesmo modelo de aprendizado para o contexto da programação automática.

Nesse capítulo apresentamos o conceito de algoritmo de estimação de distribuição e três exemplos de EDAs. Dois são exemplos de EDAs aplicados a programação automática.

O terceiro é um sistema de otimização que servirá de base para a técnica que será proposta nesse trabalho e que será exposta no próximo capítulo.

CAPÍTULO 4

PROGRAMAÇÃO AUTOMÁTICA BAYESIANA

Nesse capítulo serão introduzidas as técnicas e modelos usados para compor a Programação Automática Bayesiana, que nada mais é do que a união das diferentes técnicas introduzidas nos capítulos anteriores, de forma a compor uma ferramenta para a indução de programas através de um modelo de distribuição probabilística. No decorrer das explicações de cada tópico serão abordadas modificações introduzidas para compor a ferramenta.

4.1 Algoritmo Principal

Esta seção descreverá o algoritmo principal da PAB, que basicamente imita o algoritmo do BOA.

Tal como o BOA, a PAB usa uma rede Bayesiana para estimar a distributição de probabilidade conjunta de soluções promissoras. Esta distribuição é usada para gerar novos candidatos a solução. O BOA foi desenvolvido como um método de otimização similar a Algoritmos Genéticos. A PAB une a flexibilidade do paradigma da Programação Genética para tentar estender o poder de descrição da rede Bayesiana introduzido no BOA para a criação automática de programas e não somente para a otimização.

A representação de soluções da Programação Genética usada na PAB possui ainda vantagens sobre a representação de solução por vetor usada no BOA, pois ela permite a utilização da hierarquia implícita na representação do programa por árvore da PG como informação **a priori** para o algoritmo K2. Mais ainda, o uso de uma gramática para reger a construção dos cromossomos ajuda a restringir o espaço de busca.

O algoritmo inicialmente gera uma população inicial usando um método de geração uniformemente aleatório. Os cromossomos são avaliados por uma função de aptidão e selecionados da população atual usando um método de seleção (elitismo, torneio, etc). O

melhor cromossomo selecionado é repetido no conjunto de cromossomos selecionados de modo a ocupar 10% do tamanho desse conjunto. Essa medida foi escolhida empiricamente e tem como objetivo reforçar o aprendizado, para acelerar a convergência do algoritmo para um ótimo.

Uma rede Bayesiana que descreva o conjunto de cromossomos selecionados é aprendida usando o algoritmo K2 e restrições. Novos cromossomos são gerados usando a distribuição de probabilidade conjunta codificada pela rede. E em seguida, podem vir a sofrer mutação. O conjunto dos genes selecionados é colocado junto com os novos cromossomos, formando uma nova população que substitui completamente a antiga. O processo é repetido até que o critério de parada seja satisfeito. O pseudocóndigo da PAB é mostrado no algoritmo 4.

Algoritmo 4 Algoritmo Principal da PAB

1: $t = 0$.
2: Gera uniformimente aleatório a população inicial P(0).
3: Seleciona um conjunto de cromossomos promissores $S(t)$ de $P(t)$.
4: Constrói a rede Bayesiana B usando uma métrica e restrições.
5: Gera um conjunto de novos cromossomos $O(t)$ de acordo com a distribuição conjunta codificada por B e uma gramática G.
6: Aplica mutação sobre $P(t)$.
7: Cria uma nova população $P(t+1)$ substituindo alguns cromossomos de $P(t)$ com $O(t)$.
8: $t = t + 1$.
9: Se o critério de parada não for satisfeito, vá para o passo 3.

4.2 Codificação do Indivíduo

Nesta seção será introduzida a representação de cromossomo da PAB. Em paralelo, compararemos com o modelo no qual ela foi baseada. Também exporemos os fatores que levaram a essas modificações, de forma que a representação se tornasse adequada ao

formato de dados usado pelo K2.

Como definido na seção 3.3, uma rede Bayesiana pode ser representada pelo par $(\mathbf{X}, G)$, onde $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ é um conjunto de variáveis aleatórias e G é um grafo.

Para fazer uso de uma rede Bayesiana com o objetivo de descrever a distribuição de probabilidade entre um conjunto de cromossomos, é necessário definir quais características dos cromossomos serão modeladas pelas variáveis aleatórias e como estas características serão relacionadas com quais variáveis. Além disso as suposições assumidas pelo K2 devem ser seguidas.

Em princípio esperavamos empregar a representação do cromossomo binário usada na GE, pois ele atende a todos os requisitos básicos. Além disso essa representação é equivalente ao usado pelo BOA, nosso algoritmo de referência para evolução probabilística. Assim como no BOA, cada gene do cromossomo estaria relacionado com uma variável aleatória. No entanto, a representação por cromossomo binário se mostrou inviável, dada as características desse modelo: cromossomo sem atributos que possãm ser associados adequadamente à variáveis aleatórias e seu grande tamanho.

Consideramos o cromossomo de GE inadequado por ser uma pré-representação que deve ser alterada antes de ser efetivamente usada como informação. Ao contrário do cromossomo usado no BOA, o valor de um gene na GE não tem significado semântico direto que possa ser definido como variável aleatória e adequadamente explorado pelo uso de uma rede Bayesiana.

O tamanho do cromossomo no formato binário também é um limitante, dada a ordem de complexidade do K2, que é dependente do número de variáveis aleatórias.

Por causa dessa limitação da representação binária, é preferível usar um vetor de inteiros, que nada mais é que o resultado da conversão dos agrupamentos de números binários para a base decimal. Essa representação mantem todas as características desejáveis do cromossomo como proposto na GE e atende aos requisitos impostos pelo K2. Assim, os cromossomos seriam criados como vetores de inteiros, descartando a fase binária. Nesse modelo de representação as variáveis aleatórias compreenderão todos os possíveis valores que um determinado gene pode vir a ter.

Considerando essa representação, vamos analisar as conseqüências de usar todos os possíveis valores inteiros de um gene como variáveis aleatórias.

É necessário lembrar que, como visto na seção 2.8 sobre a GE, o uso de um inteiro para escolher a regra de formação derivada de um terminal, pede um número grande ou um multiplo comum para evitar o problema da distribuição probabilística irregular. Com o modelo atual já não há a limitação do número obrigatoriamente ser uma potência de dois, qualquer número pode ser escolhido.

Dada a ordem de complexidade do K2, $O(m\ u^2\ n^2\ r)$, é possível notar que o número de possíveis eventos r influi na complexidade do algoritmo. Logo é interessante que o valor máximo de um gene seja o mínimo multiplo comum do número de produções para cada não-terminal da gramática. Essa condição pode ser difícil de se obter em casos onde a gramática é extensa.

Outro fator que conta contra esse modelo, é a associação indireta entre o gene e a regra de formação que ele escolhe. Como o mesmo valor de um gene seleciona diferentes regras de produção dependendo de qual não-terminal ele vai traduzir, o significado desse valor fica dependente do passo de produção em que ele é aplicado. Essa informação sozinha é ambígua e pode gerar distorções nas probabilidades descritas nas TPC. Por exemplo: o evento e pode ser relacionado à operação de soma em um cromossomo e à operação seno em outro, mas a TPC vai descrever a probabilidade de e ocorrer levando em conta simplesmente a sua freqüência, independente da operação.

Para evitar essa ambigüidade, cada regra de formação será identificada por um valor único. Por padrão, o valor 0 é atribuído à primeira regra; às seguintes formações é atribuído o valor da regra antecessora mais um. Seguindo esse princípio, a gramática da figura 2.9 agora é enumerada como mostrado na figura 4.2.

N={expr,boper,var}, T={X,Y,+,-,%,*, cos, log, exp} e S={expr}

$$
P = \left\{
\begin{array}{lcll}
 & & & \text{Enumeração} \\
< \text{expr} > & ::= & < \text{expr} > \; < \text{boper} > \; < \text{expr} > & (0) \\
 & | & < \text{uoper} > \; < \text{expr} > & (1) \\
 & | & \text{X} & (2) \\
< \text{boper} > & ::= & + & (3) \\
 & | & - & (4) \\
 & | & * & (5) \\
 & | & / & (6) \\
< \text{uoper} > & ::= & \text{sen} & (7) \\
 & | & \cos & (8) \\
 & | & \log & (9) \\
 & | & \exp & (10)
\end{array}
\right.
$$

Figura 4.1: As regras de produção da gramática apresentada na figura 2.9, seção 2.8, agora enumeradas seqüencialmente.

A modificação da enumeração faz com que a escolha de uma regra de formação segundo o valor retornado pela função 2.1 não seja mais aplicável. Ao invés disso cada gene codificará a seleção de uma única regra de formação. Logo, os cromossomos não podem mais ser criados aleatoriamente. Essa restrição é facilmente satisfeita se a árvore de sintaxe concreta de uma possível solução for mapeada em um vetor com as características já discutidas.

Para produzir um vetor com essas características basta construir a árvore de sintaxe concreta usando os números atribuídos a cada regra de formação, caminhar pela árvore

inorder a partir da raiz e concatenar o valor dos nós em um vetor a cada passo. A figura 4.2 mostra uma árvore de sintaxe concreta e o vetor originado dela.

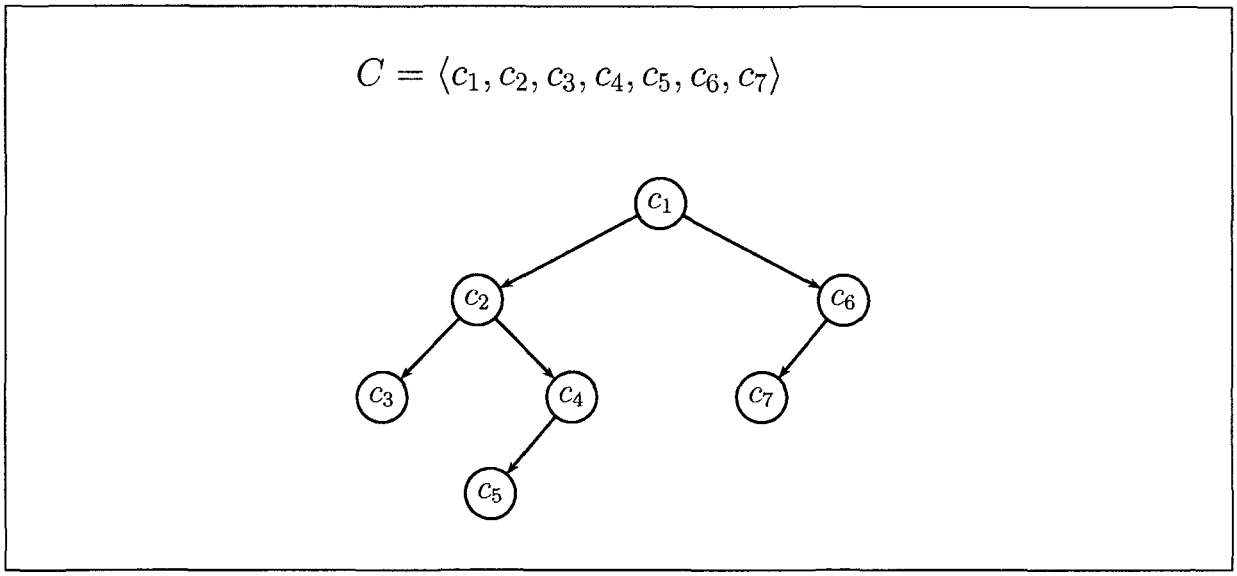


Figura 4.2: Uma árvore de sintaxe concreta e o vetor derivado dela.

O vetor pode ser construído diretamente da gramática, sem a necessidade se construir a árvore de sintaxe concreta.

Agora que já possuimos uma representação para o cromossomo e foi definido qual informação do cromossomo está modelada pelas variáveis aleatórias, é necessário definir qual gene está relacionado com qual variável. Na PAB, a relação entre um cromossomo e as variáveis aleatórias se dá através da seguinte definição: $\forall\ c \in C_j,\ c \in X_j$ tal que $X_j \in \mathbf{X}$ e $1 \leq j \leq n$. Onde $\mathbf{C} = \langle C_1, C_2, ..., C_n \rangle$, $\mathbf{C}$ é um cromossomo, C_i é o conjunto de todos os valores assumidos pelo gene na posição i em $\mathbf{C}$ e n é o número máximo de genes de $\mathbf{C}$.

O mapeamento da árvore de sintaxe concreta num vetor de inteiros feito **inorder** tem como grande vantagem o fato de manter a hierarquia dos nós da árvore no vetor. A hierarquia no cromossomo é transportada para as variáveis aleatórias e tem como conseqüência que $\forall\ X_i\ e\ X_j \in \mathbf{X}$, X_i só poderá ser pai de X_j se $i > j$. Essa informação pode ser usada pelo K2 para acelerar a busca pela melhor estrutura.

A figura 4.3 mostra um exemplo de duas árvores de sintaxe concreta e seus respectivos

cromossomos, A e B, associados com variáveis aleatórias segundo os critérios definidos anteriormente.

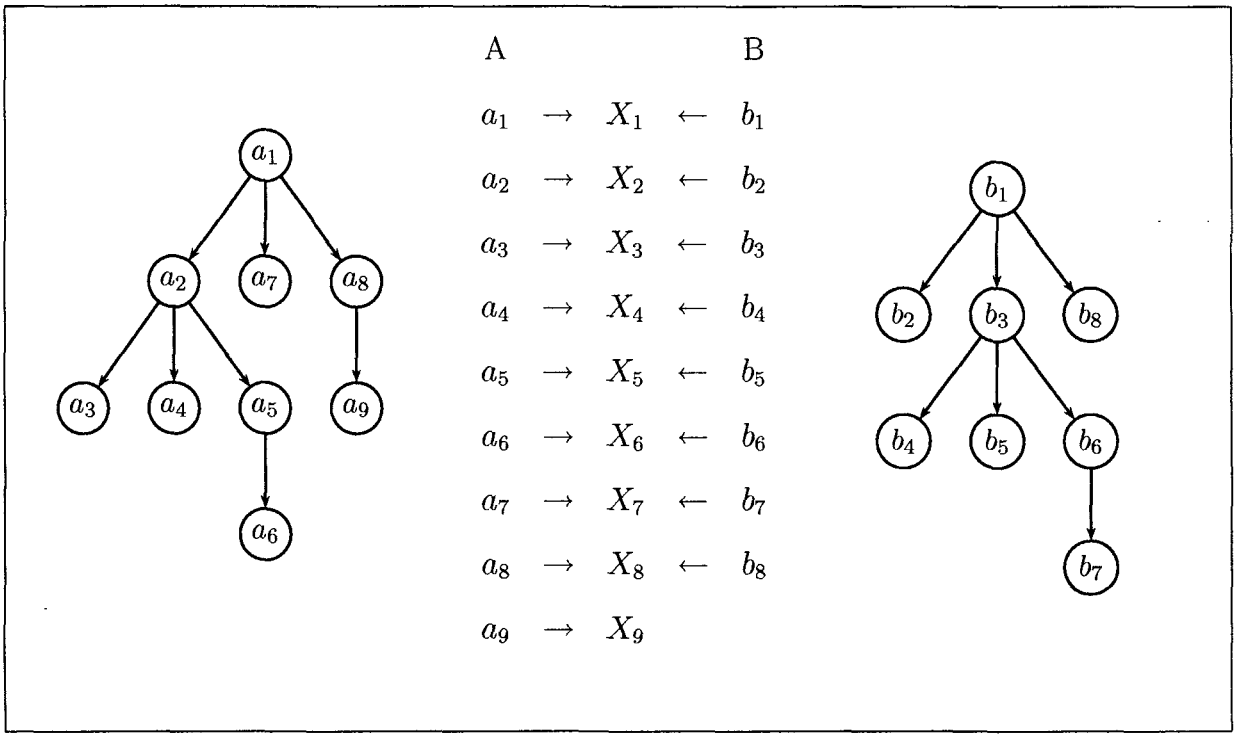


Figura 4.3: Duas árvores de sintaxe concreta diferentes tem seus nós a_n e b_n relacionados à mesma variável x_n de uma rede Bayesiana ao caminhar nas árvores **inorder**.

Para serem usados como dados de treinamento para o K2, os cromossomos devem ainda respeitar a suposição 3 dada em seção 3.3.1. Em outras palavras, é necessário que todos cromossomos tenham tamanho igual a $|\ \mathbf{X}\ |$.

De forma a cumprir com esse requisito, cromossomos que tenham tamanho menor que o necessário são completados com números aleatórios selecionados entre os inteiros atribuidos às regras de formação da gramática. A inserção de valores aleatórios para completar o cromossomo tem como efeito colateral a introdução de ruído na rede Bayesiana que será aprendida a partir desses dados.

4.3 Geração do Cromossomo

Nessa seção os dois métodos para a geração de novos cromossomos usados na PAB serão introduzidos.

A composição de populações com alta diversidade é uma questão chave em AE. Uma hipótese comum é que a alta diversidade é importante para evitar convergência prematura e para escapar de ótimos locais [39]. Na PAB, um cromossomo é representado como uma tupla C criada aleatóriamente ou usando a influência de uma rede Bayesiana.

Uma forma fácil para compor um cromossomo é selecionar aleatóriamente uma regra de formação de uma gramática quando for necessário traduzir um não-terminal. Esta "solução obvia" tem dois pontos desfavoráveis. O primeiro é que ele pode gerar cromossomos extremamente longos. O segundo é que nem todos os possíveis cromossomos de tamanho máximo $\mid C_{max} \mid$ têm a mesma probabilidade de serem gerados. Exemplo: dada a gramática

$$S \longrightarrow a \mid A;$$
$$A \longrightarrow b \mid c;$$

, ela irá gerar a com probabilidade $\frac{1}{2}$ enquanto que b e c serão gerados com probabilidade $\frac{1}{4}$. A estrutura da gramática também influencia a probabilidade. Duas gramáticas distintas podem gerar o mesmo cromossomo com diferentes probabilidades. [26].

4.3.1 Geração dos Cromossomos da População Inicial

Para evitar os problemas listados na seção 4.3, foi usado um método recursivo com a capacidade de gerar cromossomos aleatórios uniformemente distribuidos sobre o espaço de busca.

Dentre todos os possíveis cromossomos de um determinado tamanho que possam ser gerados a partir de uma gramática, o algoritmo gera todos com probabilidade igual.

Dado um determinado tamanho de cromossomo e uma gramática, o algoritmo gera cromossomos com probabilidade igual para todos os possíveis.

O algoritmo tem uma etapa de pré-processamento onde o número de todos possíveis cromossomos de um tamanho fixo derivados de uma gramática livre de contexto pode ser obtido. Ao contar o número de todos possíveis cromossomos com tamanhos entre zero e o tamanho máximo permitido para um cromossomo é possível descobrir a distribuição T, que descreve o número de todos os cromossomos possíveis por tamanho. Essa distribuição pode ser usada para criar uma população de cromossomos uniformimente distribuída. Para informações mais especificas sobre o algoritmo consultar [26].

Para produzir um cromossomo selecionado uniformemente aleatório, primeiro devemos escolher aleatóriamente seu tamanho segundo T. Com o tamanho definido, um cromossomo é construído aleatóriamente usando o método recursivo.

4.3.2 Geração da População Usando Uma Rede Bayesiana

Todos os cromossomos gerados após a população inicial são construídos usando o processo de mapeamento. Este processo é inspirado no processo de criação de um cromossomo proposto em GE como introduzido na seção 2.8. Porem, modificações foram introduzidas dado os diferentes objetivos do processo de mapeamento na PAB e na GE. Foram feitas modificações na:

- Representação do cromossomo - De uma representação por vetor de número binários passou-se para um vetor de números inteiros.

- Significado semântico do cromossomo - De um cromossomo composto por genes gerados de forma arbitrária passou-se para uma composição onde foi adicionada significado semântico a cada gene, através da modificação do método de geração do cromossomo.

Na GE, os cromossomos são criados como vetores de números binários selecionados aleatóriamente. Nesse contexto a gramática tem a função de decodificar o cromossomo. Na PAB, o cromossomo é um subproduto da gramática, pois ele nada mais é que uma forma de representação de uma árvore de sintaxe concreta da gramática.

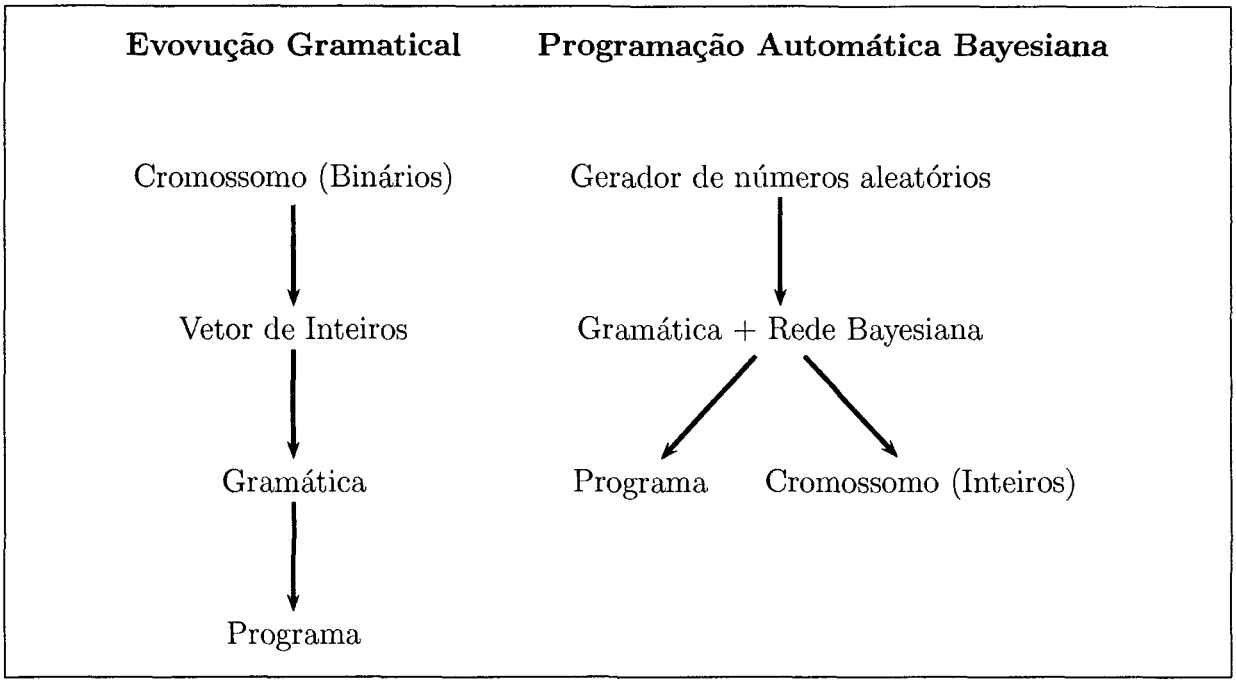


Figura 4.4: Analogia entre o modelo de composição de um cromossomo na Evolução Gramatical e na Programação Automática Bayesiana.

A modificação da semântica dos cromossomos nos dois algoritmos e a introdução da informação fornecida pela rede Bayesiana pede um método de criação distinto, apesar de algumas semelhanças entre os dois métodos.

O principal objetivo ao criar um cromossomo é usar a informação da rede Bayesiana e garantir que o cromossomo é válido.

Um comossomo é considerado válido se ele puder ser traduzido em uma árvore de sintaxe concreta contendo apenas terminais e tiver tamanho menor ou igual a $\mid C_{max} \mid$.

A seguir veremos que nem sempre é possível atingir os dois objetivos simultaneamente.

O algoritmo de criação de cromossomo usando a rede Bayesiana, cujo pseudocódigo está representado no algoritmo 5, é muito semelhante ao processo de produção de uma sentença usando uma gramática livre de contexto. Ao final do processo teremos o cromossomo e a sentença que ele codifica, que nada mais é que uma possível solução para o problema.

O algoritmo começa criando um vetor I, inicialmente vazio, e concatenando nele o símbolo de inicialização da gramática. I conterá as formas intermediárias da sentença

e no final do processo conterá o programa codificado pelo cromossomo C. Em seguida, I é varrido do gene 1 até seu último gene em busca de um não-terminal que deva ser traduzido por uma das regras da gramática.

A escolha de uma das regras de formação é feita por uma algoritmo de seleção do tipo roleta [35], usando a distribuição de probabilidade descrita pela TPC da variável aleatória referente ao gene que receberá o valor da regra de formação que será selecionada.

Na seleção por roleta uma classe tem a chance de ser selecionada proporcional a seu valor com relação ao todo, que é a soma do valor de todas as classes. Efetivamente, o algoritmo de roleta recebe como parametro a rede Bayesiana treinada B e a posição do gene que está sendo selecionado, i.

De posse de B, o algoritmo a consulta para verificar se X_i possui algum parente, gerando duas possíveis situações.

- X_i possui parentes: Nesse caso, os valores em C relacionados as variáveis aleatórias que são parentes de X_I são recuperados e usados para selecionar as distribuições de probabilidade dos eventos descritos na TPC de X_i.

- X_i não possui parentes: Nesse caso a tabela simplesmente retorna a distribuição de probabilidade de todos os eventos da TPC de X_i.

Já de posse da distribuição de probabilidade de todos os possíveis eventos descritos em X_i, o algoritmo de seleção usa um número real gerado aleatóriamente e normalizado para escolher um entre os eventos de distribuição que possam traduzir o não-terminal em questão. A chance de cada evento ser selecionado é proporcional a sua probabilidade descrita na distribuição.

Após escolher o evento, ele é atribuído ao gene na posição i de C. O não-terminal em I é removido e substituído pelo lado esquerdo da produção escolhida. O vetor I é novamente varrido do gene 1 para o final em busca de um não-terminal a ser traduzido e o processo de seleção recomeça.

O algoritmo 5 mostra o pseudo-código do processo de mapeamento de um cromossomo C.

Algoritmo 5 Algoritmo de criação de um cromossomo

1: $i = 1$.
2: I=$S \in G$. I é um vetor auxiliar que armazena não-terminais e terminais.
3: **enquanto** I contêm não-terminais e $i \leq$ tamanho máximo de um cromossomo **faça**
4: r=Número aleatório normalizado.
5: c=roleta(r,TPC de X_i). X_i é a i-ésima variável aleatória de B.
6: Concatena c em C.
7: Remove o primeiro não-terminal em I e o substitui pelo lado direito da regra de formação escolhida.
8: $i = i + 1$.
9: **fim enquanto**
10: **se** I não possui não-terminais **então**
11: Retorna C.
12: **senão**
13: Retorna FALHA.
14: **fim se**

O processo de recuperação das distribuições de probabilidade da TPC de uma variável aleatória que tenha parentes nem sempre acontece de forma ideal como descrito anteriormente.

Eventualmente, a TPC de variável aleatória não contém uma entrada descrevendo a probabilidade condicional de seus eventos e os eventos dos seus parentes, que já estão fixos no cromossomo. Há ainda uma situação onde a TPC da variável aleatória não possui nenhum evento que possa ser usado dado o não-terminal a ser traduzido.

Esses problemas acontecem por que a rede Bayesiana não descreve todas as possíveis relações de condição entre os genes, mas somente aquelas que ultrapassam um certo limite que é definido pelo algoritmo de busca da rede, e também porque a probabilidade Bayesiana considera apenas os valores observados nos cromossomos usados para aprender a rede.

A seguir descrevemos um exemplo que apresenta esses dois problemas. Cinco cromossomos, apresentados na tabela 4.1, com no máximo 15 genes foram escolhidos para o K2 aprender a estrutura da rede. A figura 4.3.2 mostra a gramática usada para criar os cromossomos. O K2 retornou a estrutura descrita na figura 4.6 e a tabela 4.2 apresenta

os eventos presentes nos nós 3, 4 e 5.

N={expr,boper,uoper,var}, T={X,1,+,-,%,*sen,cos,log,exp} e S={expr}

$$
P = \begin{cases}
 & & & & & \text{Enumeração} \\
< \text{expr} > & ::= & < \text{expr} > & < \text{expr} > & < \text{boper} > & (0) \\
 & | & < \text{expr} > & < \text{uoper} > & & (1) \\
 & | & < \text{var} > & & & (2) \\
 & | & < \text{var} > & & & (3) \\
< \text{boper} > & ::= & + & & & (4) \\
 & | & - & & & (5) \\
 & | & * & & & (6) \\
 & | & / & & & (7) \\
< \text{uoper} > & ::= & \text{sen} & & & (8) \\
 & | & \text{cos} & & & (9) \\
 & | & \text{log} & & & (10) \\
 & | & \text{exp} & & & (11) \\
< \text{var} > & ::= & \text{X} & & & (12) \\
 & | & 1 & & & (13)
\end{cases}
$$

Figura 4.5: Gramática usada na composição do exemplo do problema de descrição da rede Bayesiana.

Cromossomo 1	1	1	1	1	1	3	13	11	10	10	10	10	3	13	13
Cromossomo 2	0	0	2	12	1	3	13	9	5	3	12	7	6	4	5
Cromossomo 3	1	1	0	2	13	1	3	12	9	7	11	9	4	0	4
Cromossomo 4	1	1	1	1	1	3	13	11	10	10	10	10	0	2	6
Cromossomo 5	1	1	1	1	1	3	13	11	10	10	10	10	13	7	3

Tabela 4.1: Cromossomos escolhidos dos quais o algoritmo K2 aprendeu a rede Bayesiana

Nó	Eventos no nó
1	0,1
2	0,1
3	1, 2, 0
4	1, 13
5	3, 1

Tabela 4.2: Eventos descritos na TPC nos nós de 1 a 5 da rede Bayesiana. Os eventos são as regras de produção que foram observados nos genes da tabela 4.1.

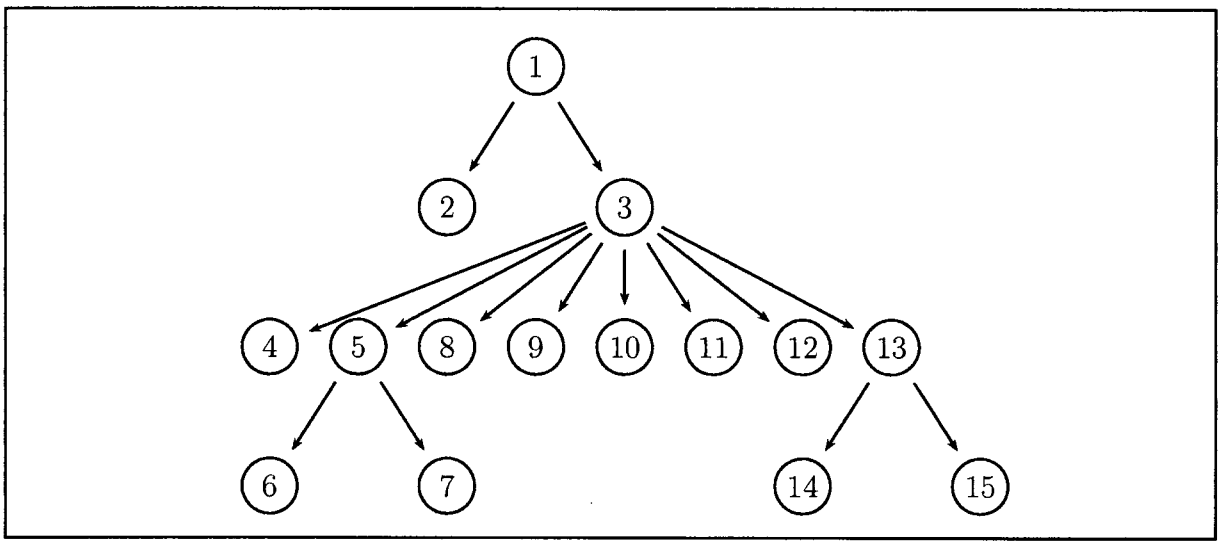


Figura 4.6: Representação gráfica da estrutura da rede Bayesiana aprendida a partir dos dados.

Caso os seguintes valores sejam atribuidos para os quatro primeiros genes de um cromossomo $C = \{1, 1, 2, 12\}$, a gramática pede que o próximo valor seja 8, 9, 10 ou 11. No entanto o nó 5 não possui nenhuma dessas possibilidades. A figura 4.7 mostra os passos de produção do cromossomo e do programa que ele codifica.

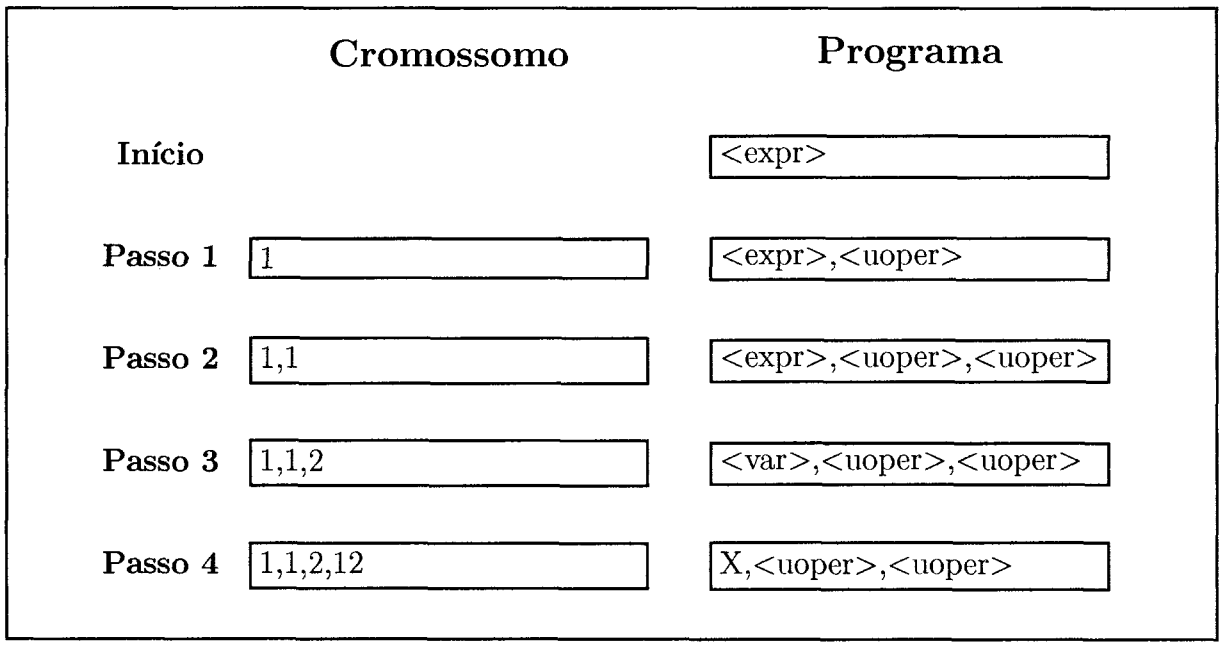


Figura 4.7: Passos de produção de um cromossomo que não pode ser contruído dada rede Bayesiana da figura 4.6 e o programa que ele codificou.

Para transpor essa limitação, as restrições impostas pela rede Bayesiana são desconsideradas de modo que o cromossomo seja construído. Caso um evento necessário não esteja presente nos nós pais de um nó que modela as probabilidades de um gene que deve ser preenchido, ele é desconsiderado e a TPC é consultada como se o nó não tivesse pai. Caso os eventos que a gramática exige não estejam presentes na TPC da variável aleatória, eles são criados temporariamente com probabilidade uniformemente distribuída de serem selecionados.

O processo de composição de um cromossomo usando uma gramática como aplicado no mapeamento tem ainda a tendência a gerar cromossomos muito longos ou muito pequenos, como exposto na subseção 2.7.2.

Por causa desse efeito foi necessário usar uma restrição para controlar o tamanho mínimo e máximo de um cromossomo criado de forma a obter-se bons resultados. Essa

restrição é feita através de uma constante δ definida pelo usuário.

Após sua criação, um cromossomo deve ter tamanho $m - \delta \leq \mid Y \mid \leq m + \delta$, onde m é a média aritmética do tamanho de todos os cromossomos selecionados como dados para o aprendizado da rede. Caso essa restrição seja desrespeitada o cromossomo é considerado inválido, ele é descartado e outro gerado em seu lugar.

4.4 Mutação

Nesta seção introduziremos uma operação de mutação que considera a homogeneidade da população.

Após algumas gerações a população pode se tornar homogênea, já que características de cromossomos de gerações anteriores são perdidas e indivíduos selecionados tendem a ser reproduzidos mais que os outros. Isso pode levar a convergência para um ótimo local. A PAB usa uma operação de mutação para tentar evitar ótimos locais.

Na mutação tradicional aplicada à PG, uma porção de um indivíduo (uma sub-árvore ou uma folha) é substituída por uma nova, criada normalmente de forma aleatória. Cada nó do indivíduo é testado para verificar se ele sofrerá mutação. Todos os nós possuem a mesma probabilidade de sofrer mutação.

A abordagem tradicional não se mostrou eficiente o bastante para evitar a estagnação da evolução devido a um ótimo local. Essa ineficiência se deve a diferença entre o modo como um Algoritmo Evolucionário, da qual se origina a abordagem tradicional, e PAB compõe uma nova população.

Como em PAB a população é totalmente substituida por uma nova, criada a partir de uma rede Bayesiana, a diversidade da população está sugeita a diversidade que a rede treinada pode gerar. Como a rede é treinada usando apenas uma pequena parcela da população, a diversidade da população tende a diminuir a cada geração. Essa diminuição da diversidade populacional não é tão acentuada nos Algoritmos Evolucionários, que usam operadores genéticos para produzir uma nova população.

Para contornar essa situação, uma operação de mutação que responde à homogeneidade da população foi criada. Segundo essa proposta, quanto maior a similaridade

entre um cromossomo produzido e a população anterior, maior a probabilidade dele sofrer mutação.

O grau de semelhança entre um cromossomo criado e a população é calculado ao se comparar a freqüência de cada gene desse cromossomo com a sua freqüência em uma tabela contendo uma amostra dos cromossomos da população anterior, selecionados aleatoriamente.

De forma a contar a freqüência de um evento na tabela T, que contêm os cromossomos selecionados aleatoriamente, uma função f é definida como segue:

$$f(x,y,z) = \begin{cases} 1 & x = T_{y,z}, \\ 0 & \text{caso contrário.} \end{cases} \tag{4.1}$$

Um cromossomo C sofrerá mutação se:

$$\sum_{j=1}^{|C|} \sum_{i=1}^{n} \frac{f(c_j,i,j)}{n} > \frac{|\ C\ |}{\rho} \tag{4.2}$$

Onde ρ é uma constante de controle definida pelo usuário, n é o número de cromossomos (linhas) em T. Quanto maior o valor de ρ, menor será o fator de similaridade necessário entre o cromossomo testado e a amostra, para que o cromossomo sofra mutação.

Uma constante de mutação μ, também definida pelo usuário, controla a probabilidade de cada gene de um cromossomo sofrer mutação. A seleção do ponto de mutação ocorre como segue: dado um cromossomo C e μ, a mutação ocorrerá no gene $c \in C$ caso a inequação 4.3 seja satisfeita.

$$\frac{\mu}{|\ C\ |} \geq r \tag{4.3}$$

Onde r é um número normalizado, selecionado aleatóriamente entre os reais. Para cada c testado um novo r é selecionado.

Se a inequação é satisfeita, um não-terminal é procurado a partir do ponto de mutação até o último gene. Caso nenhum não-terminal seja encontrado, outro ponto de mutação é selecionado a partir do primeiro gene de c. Caso contrário, a representação por árvore

de sintaxe concreta de C é reconstruída e toda a sub-árvore que deriva do gene mutado é removida.

Uma nova sub-árvore é gerada usando o método de geração uniformemente aleatório apresentado na seção 4.3.1. O tamanho da nova sub-árvore é selecionado aleatóriamente entre todos os possíveis tamanhos cuja soma com $\mid C \mid$ menos o tamanho da sub-árvore removida é menor que o tamanho máximo permitido a um cromossomo. A raiz da sub-árvore criada é um não-terminal do mesmo tipo que o removido, o que garante que a expressão mutada é válida.

Após a modificação o cromossomo é novamente convertido na sua forma de vetor e inserido na nova população.

4.5 Representação Esquemática da Ferramenta Implementada

Para validar a técnica proposta, uma versão da PAB em C++ foi implementada. A figura 4.8 mostra uma representação esquemática da implementação.

- **Gerador de Números Aleatórios** Para obter uma boa qualidade nos números aleatórios, foi usando o algoritmo MersenneTwister da biblioteca para computação científica do projeto GNU (GNU Scientific Library) [13].

- **Gerador Uniformemente Aleatório** Esse modulo é responsavel por gerar cromossomos de forma uniformemente aleatória. Ele recebe uma especificação da gramática, produz os cromossomos da geração inicial e gera subexpressões usadas no modulo de mutação.

- **Gramática** Contem a definição de uma gramática livre de contexto.

- **Gerador Bayesiano de Cromossomos** Usa as informações da gramática e a distribuição de probabilidades modelada pela rede Bayesiana para produzir novos cromossomos.

- **População** Armazena os cromossomos.

- **Mutação** Aplica o operador de mutação nos cromossomos que ele recebe e os insere na População.
- **Seleção** Seleciona os melhores indivíduos da população por elitismo.
- **Avaliador de Aptidão** Usa um interpretador para avaliar cada indivíduo e lhes atribui um valor de aptidão.
- **K2** Recebe um conjunto de cromossomos e aprende a rede Bayesiana que descreve distribuição de probabilidade desses cromossomos. Tabelas de probabilidade condicional são usadas para armazenar as probabilidades de cada variável aleatória.

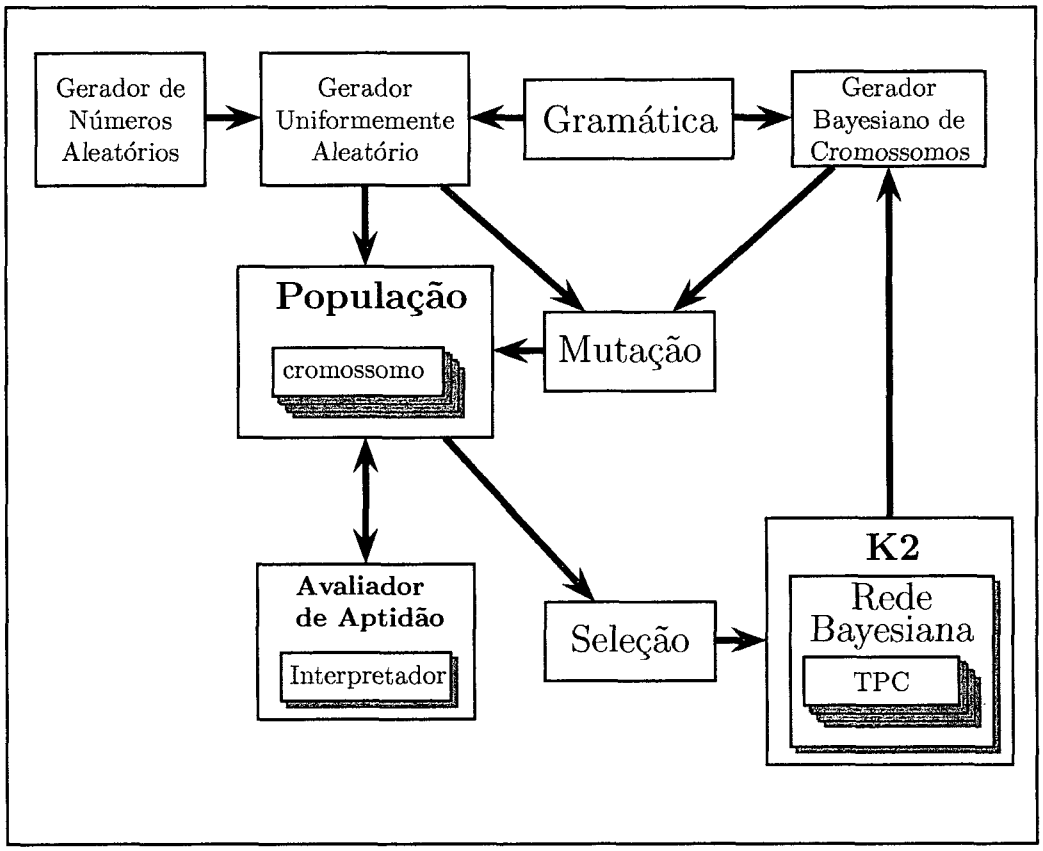


Figura 4.8: Diagrama com os componentes da PAB implementado.

A ferramenta inicialmente instancia o gerador de números aleatórios com uma semente fornecida pelo usuário. Em seguida o módulo de geração de cromossomos uniformemente aleatório gera a população inicial seguindo a definição da gramática, formando uma população de cromossomos. Após serem criados, os cromossomos considerados válidos são

imediatamente avalidados através de um interpretador. Um algoritmo de seleção é então usado para escolher os indivíduos por elitismo ou por torneio. O conjunto de cromossomos selecionados é usado pelo K2 para aprender uma rede Bayesiana e para contruir as tabelas de probabilidade condicional que são usadas, juntamente com a gramática, pelo gerador Bayesiano de cromossomos para gerar uma nova população.

Nesse capítulo foram apresentadas as principais contribuições desse trabalho. Foram abordadas todas as estruturas e técnicas usadas para adaptar a representação de indivíduo da PG de forma a cumprir as restrições do algoritmos K2. Também propomos um operador de mutação. No capítulo seguinte apresentaremos as configurações de três problemas e o desempenho da técnica proposta que será comparada com a PG tradicional.

CAPÍTULO 5

EXPERIMENTOS

Para avaliar sua performance, a PAB foi aplicada à dois tradicionais problemas de programação genética: o problema de regressão simbólica e o problema da formiga artificial.

Em todos os testes, duas versões da PAB, uma com a mutação proposta e outra com a mutação tradicional, são comparadas com a programação genética tradicional. Os métodos foram comparados com relação a sua freqüência de sucessos acumulados e com relação a mediana do valor de adaptividade durante 50 gerações, em 100 execuções independentes.

Para os testes, foi usada uma versão da PAB como descrita na seção 4.5. A ferramenta de programação genética tradicional usada foi a Lilgp 1.1 [34].

5.1 Configurações

5.1.1 Regressão Simbólica

No problema de regressão simbólica, um conjunto de valores entrada e saída são providos. O objetivo é encontrar uma função que produza os valores de saída ao receber como parâmetro os valores de entrada. Foram realizados experimentos com duas equações:

Equação 1

A função objetivo do primeiro experimento de regressão simbólica está descrita na equação 5.1:

$$f(x, y) = x^4 + 2xy + y^4 \tag{5.1}$$

Os dados de entrada são compostos por dois conjuntos de vinte números reais selecio-

nados aleatoriamente entre a faixa $[-10...10]$. A função de aptidão é a soma do erro nos vinte casos de teste.

A gramática usada pela PAB foi:

N={expr,boper,var}, T={X,Y,+,-,%,*} e S={expr}

$$
P = \begin{cases}
< \text{expr} >::=< \text{expr} >< \text{boper} >< \text{expr} > \\
\qquad |(< \text{expr} >< \text{boper} >< \text{expr} >) \\
\qquad |< \text{var} > \\
< \text{boper} >::= + \mid - \mid * \mid \% \\
< \text{var} >::= X \mid Y
\end{cases}
$$

Figura 5.1: Gramática usada pela PAB no primeiro problema de regressão simbólica.

A operação % é uma divisão protegida que evita a divisão por zero. Sempre que uma divisão por zero acontece % retornará 1. A Tabela 5.1 contêm os parâmetros de configuração usados pela PAB e pela PG no primeiro problema de regressão simbólica.

PAB		GP	
Parametro	Valor	Parametro	Valor
Tamanho da População	500	Tamanho da População	500
Número de Gerações	50	Número de Gerações	50
ρ	2	Probabilidade de Mutação	0.01
μ	0.3	Probabilidade de Cruzamento	0.7
δ	10	Tamanho do Torneio	5
Tamanho do Cromossomo	60	Terminais Operadores	$\%, +, *, -$
		Terminais Operandos	X, Y

Tabela 5.1: Tabela com os parâmetros de configuração usados pela PAB e pela PG no primeiro problema de regressão simbólica.

Equação 2

A função objetivo para o segundo experimento com regressão simbólica está descrita na equação 5.2:

$$f(x) = x^4 + x^3 + x^2 + x \tag{5.2}$$

Os dados de entrada são compostos por um conjunto de vinte números reais selecionados aleatoriamente entre a faixa $[-1...1]$. A função de adaptidão é a soma do erro nos vinte casos de teste.

A gramática usada pela PAB foi:

N={expr, boper, var, sen, cos, log, exp}, T={X, 1, +, -, %, *} e S={expr}

$$\text{P} = \begin{cases} <\text{expr}>::=<\text{expr}><\text{boper}><\text{expr}> \\ \qquad\quad |(<\text{expr}><\text{boper}><\text{expr}>) \\ \qquad\quad |<\text{uoper}>(<\text{expr}>) \\ \qquad\quad |<\text{var}> \\ \qquad\quad |<\text{var}> \\ <\text{boper}>::= + \mid - \mid * \mid \% \\ <\text{uoper}>::= \text{sen} \mid \text{cos} \mid \text{log} \mid \text{exp} \\ <\text{var}>::= \text{X} \mid 1 \end{cases}$$

Figura 5.2: Gramática usada pela PAB no segundo problema de regressão simbólica.

A operação **exp** retorna o valor de e elevado a potência do parâmetro que ele recebe.

A operação **log** retorna o logaritmo natural do valor passado como parâmetro. Ela é protegida, assim como %. A operação log é sempre aplicada sobre o valor absoluto de um número e log retorna 0 caso o parâmetro passado seja 0.

A Tabela 5.2 contém os parâmetros de configuração usados pela PAB e pela PG no segundo problema de regressão simbólica.

O número de nós permitidos foi limitado nesse problema para uma melhor comparação entre a diversidade populacional da PAB e do Lilgp. Como efeito colateral essa modificação melhora o desempenho do Lilgp, já que limita o seu espaço de busca.

PAB		GP	
Parâmetro	Valor	Parâmetro	Valor
Tamanho da População	500	Tamanho da População	500
Número de Gerações	50	Número de Gerações	50
ρ	2	Probabilidade de Mutação	0.01
μ	0.3	Probabilidade de Cruzamento	0.7
δ	10	Tamanho do Torneio	5
Tamanho do Cromossomo	30	Terminais Operadores	$\%, +, *, -$ sen, cos, exp, log
		Terminais Operandos	$X, 1$
		Número Máximo de Nós	30

Tabela 5.2: Tabela com os parâmetros de configuração usados pela PAB e pela PG no segundo problema de regressão simbólica.

5.1.2 Formiga Artificial

O objetivo da formiga artificial é encontrar um programa que controle os movimentos de uma formiga que deve recolher tantos pedaços de comida quanto possível. A formiga é inserida em um toroidal de 32 por 32 com 91 pedaços de comida, compondo uma trilha descontínua.

A formiga é capaz de executar quatro ações: virar a esquerda, virar a direita, mover para frente, olhar a frente. A função olhar a frente permite que ela veja se há comida no quadrado imediatamente a sua frente. Cada vez que a formiga move para um quadrado onde há comida ou se o programa que a controla termina, ele é reexecutado. Cada vez que a formiga move para frente ou vira, uma unidade de uma quantidade fixa de tempo é despendida.

A execução do programa da formiga termina quando o tempo esgota ou quando todos

os pedaços de comida forem recolhidos. A função de aptidão é o número de pedaços de comida que restam no toroidal.

A gramática usado nesse problema foi:

N={expr, line,op}, T={esquerda(), direita(), move(), comida_afrente()}, S={expr}

$$
P = \begin{cases}
< \text{expr} > ::= < \text{expr} >< \text{line} > \\
\qquad\qquad | < \text{line} > \\
< \text{line} > ::= \text{Se comida\_afrente() } \{< \text{expr} >\} \text{ senão } \{< \text{expr} >\} \\
\qquad\qquad | < \text{op} > \\
< \text{op} > ::= \text{direita()} \\
\qquad\qquad | \text{esquerda()} \\
\qquad\qquad | \text{move()}
\end{cases}
$$

Figura 5.3: Gramática usada pela PAB no problema da formiga artificial.

A tabela 5.3 contêm os parâmetros de configuração usados pela BAP e pela PG no problema da formiga artificial.

PAB		GP	
Parâmetros	Valor	Parâmetro	Valor
Tamanho da População	1000	Tamanho da População	1000
Número de Gerações	50	Número de Gerações	50
ρ	3	Freqüência de Mutação	0.01
μ	0.3	Freqüência de Cruzamento	0.7
Tamanho do Cromossomo	80	Tamanho do Torneio	5
		Terminais Operandos	move(), esquerda(), direita(), comida_afrente()
		Terminais Operadores	nenhum

Tabela 5.3: Tabela com os parâmetros de configuração usados pela PAB e pela PG no problema da formiga artificial.

5.2 Resultados Experimentais

Como observado nos gráficos 5.4 e 5.8, a PAB obteve melhores resultados que a PG tradicional no primeiro problema de regressão simbólica e no problema da formiga artificial. No entanto, no segundo experimento de regressão simbólica o desempenho da PAB é bem inferior ao da PG, como mostrado no gráfico 5.6.

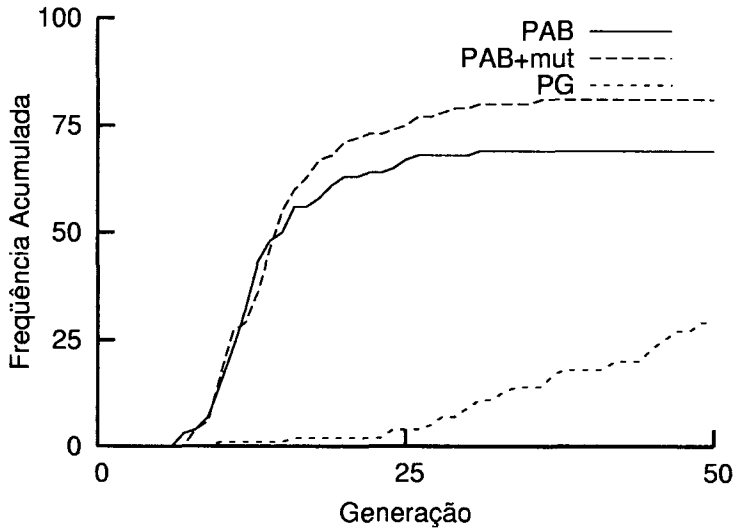


Figura 5.4: Freqüência de sucessos acumulados da PAB com a mutação proposta (PAB+mut), da PAB com a mutação tradicional (PAB) e da PG, durante as 50 gerações do primeiro problema de regressão simbólica.

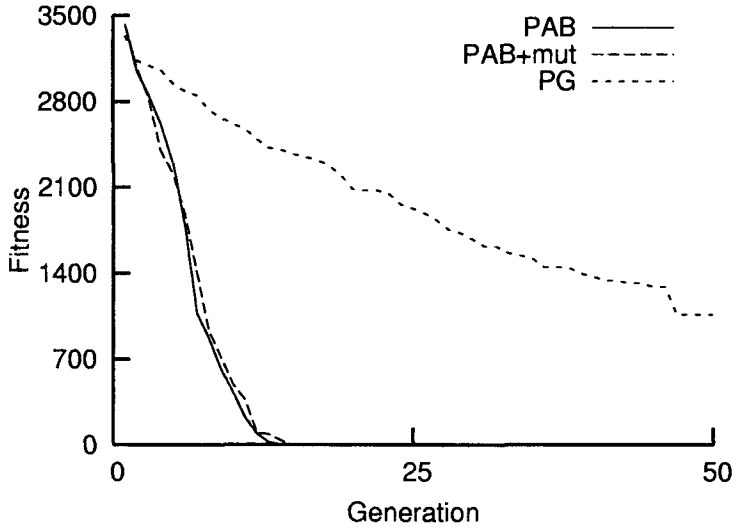


Figura 5.5: Mediana da aptidão da PAB usando a mutação proposta (PAB+mut), da PAB usando a mutação tradicional (PAB) e da PG durante as 50 gerações do primeiro problema de regressão simbólica.

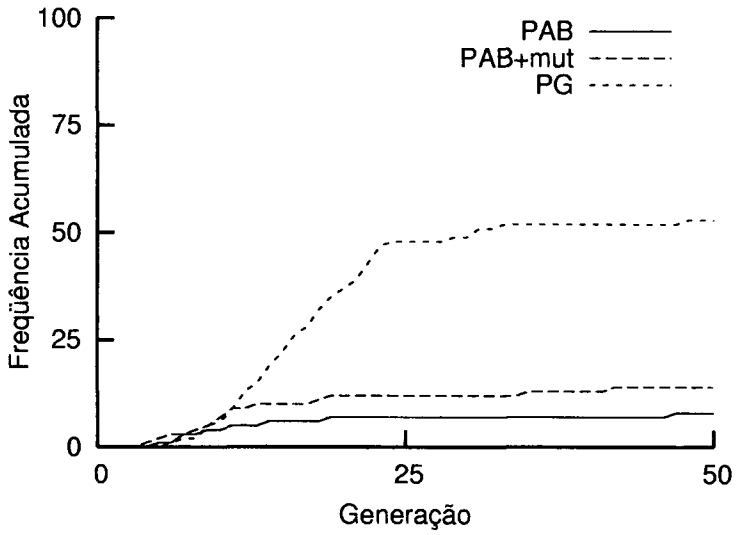


Figura 5.6: Freqüência de sucessos acumulados da PAB com a mutação proposta (PAB+mut), da PAB com a mutação tradicional (PAB) e da PG durante as 50 gerações do segundo problema de regressão simbólica.

Uma grande diferença entre a PG e a PAB é a rápida migração da segunda para o ótimo gobal, se comparado com a PG, como visto nos gráficos 5.5, 5.7 e 5.9.

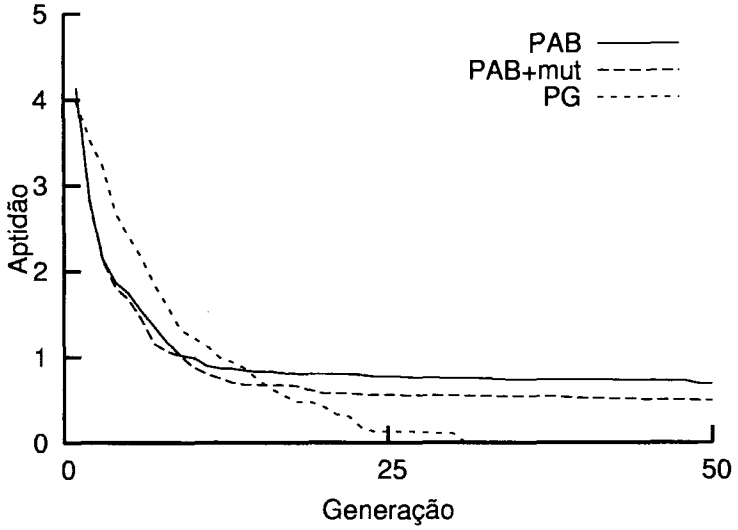


Figura 5.7: Mediana da aptidão da PAB usando a mutação proposta (PAB+mut), da PAB usando a mutação tradicional (PAB) e da PG durante as 50 gerações do segundo problema de regressão simbólica.

5.2.1 Formiga Artificial

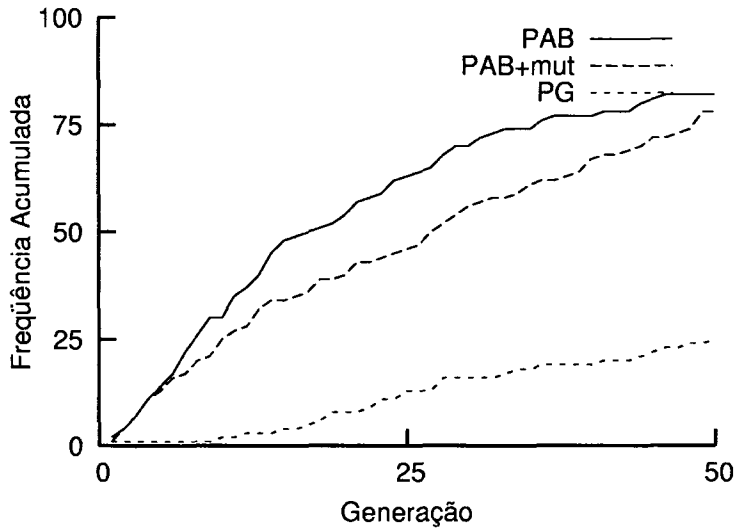


Figura 5.8: Freqüência de sucessos acumulados da PAB com a mutação proposta (PAB+mut), da PAB com a mutação tradicional (PAB) e da PG durante as 50 gerações do problema da formiga artificial

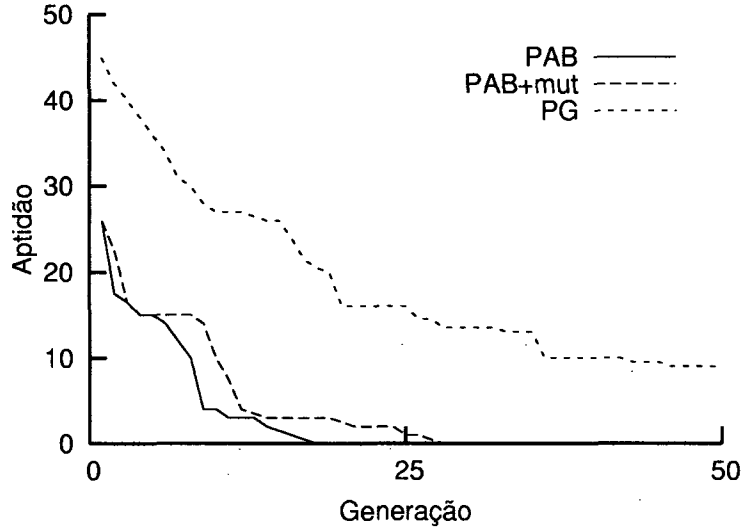


Figura 5.9: Mediana da aptidão da PAB usando a mutação proposta (PAB+mut), da PAB usando a mutação tradicional (PAB) e da PG durante as 50 gerações do problema da formiga artificial.

Apesar de não terem sido feitas comparações adequadas de performance entre a PAB e a PG, nos três problemas propostos foi observado que o tempo de execução da primeira é muito maior a que da segunda. Consideramos que a diferença de tempo de execução entre as duas abordagens é devida à ordem de complexidade do K2. Como o tempo de processamento do K2 é dependente do tamanho do cromossomo e do número de exemplos dos quais o algoritmo aprende a rede, ambos fatores tem de ser limitados para os menores valores possíveis.

Observamos também que um grande número de indivíduos são descartados pelo processo de geração de cromossomos usando a rede Bayesiana, devido à geração de cromossomos com tamanho acima ou abaixo do limite estipulado. Para medir a ordem dessa perda, compararmos o número de indivíduos gerados no segundo problema de regressão simbólica com o número de indivíduos que deveriam ser gerados caso todos eles fossem válidos.

Registramos que nas 100 execuções independentes do programa, foram gerados em média 28.394,58 cromossomos durante as 50 gerações, enquanto que o número de cromossomos que seriam criados caso todos fossem válidos é de 23.030. Logo, houve uma

perda de 18,89% dos cromossomos. O gráfico 5.10 mostra a média do número de cromossomos criados além do número mínimo nas 100 rodadas. São criados mais cromossomos inválidos nas gerações iniciais porque nessa fase os cromossomos usados para aprender a rede Bayesiana são mais heterogéneos. Logo a rede aprendida permitirá a composição de cromossomos com uma grande diversidade de tamanhos.

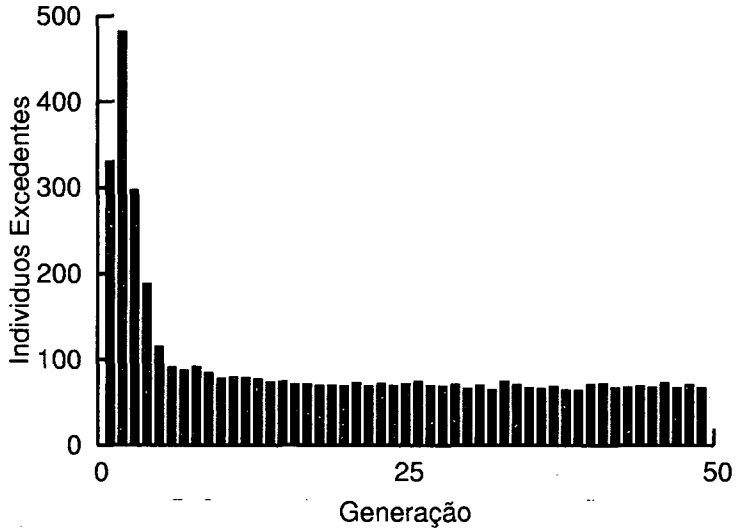


Figura 5.10: Média do número de individuos descartados por geração.

Para validar a eficiência da mutação proposta em manter a diversidade populacional, comparamos a diversidade da aptidão de todas as populações produzidas durante as 50 gerações de uma rodada da PAB e da PG no segundo problema de regressão simbólica.

Apesar das limitações do emprego da aptidão como fator diferenciador das características estruturais dos indivíduos da população, o que caracteriza sua diversidade, a aptidão é uma conseqüência dessas características. Além disso, ela é uma métrica de fácil obteção se comparada com outras, como a ordem lexicográfica. Essa escolha também será usada a seguir como argumento para justificar o desempenho inferior da PAB nesse problema de regessão simbólica.

As figuras 5.11, 5.12, 5.13 e 5.14 mostram respectivamente: uma execução da PAB com a mutação proposta, uma execução da PAB com o mutação tradicional, uma execução da PAB sem nenhuma mutação e uma execução da PG. Em nenhuma das execuções que forneceram os dados para os gráficos a solução ótima é encontrada. Todas execuções das

diferentes configurações da PAB usaram a mesma semente para inicializar o gerador de números aleatórios. Alguns valores de aptidão que apresentavam valores muito alto foram removidos dos gráficos para a melhor visualização dos resultados.

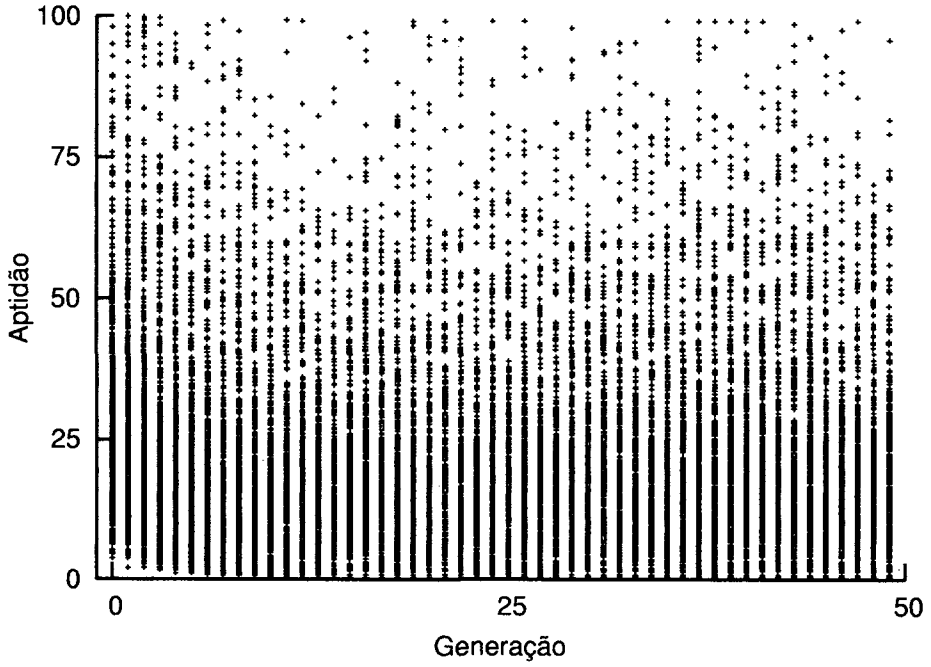


Figura 5.11: Distribuição populacional dos cromossomos gerados pela PAB usando a mutação proposta durante uma execução do segundo problema de regressão simbólica.

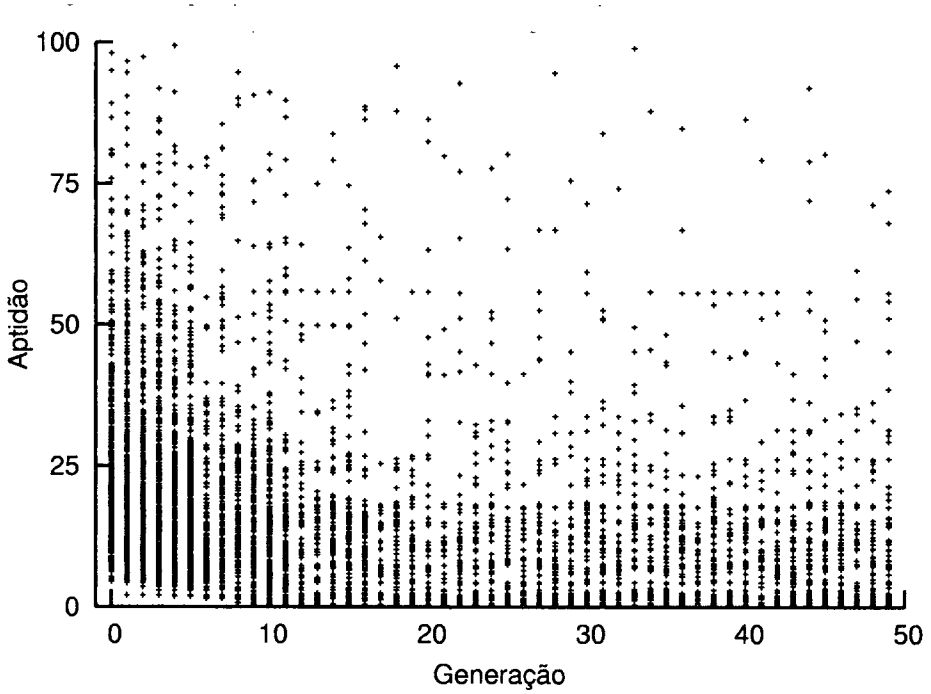


Figura 5.12: Distribuição populacional dos cromossomos gerados pela PAB usando a mutação tradicional durante uma execução do segundo problema de regressão simbólica.

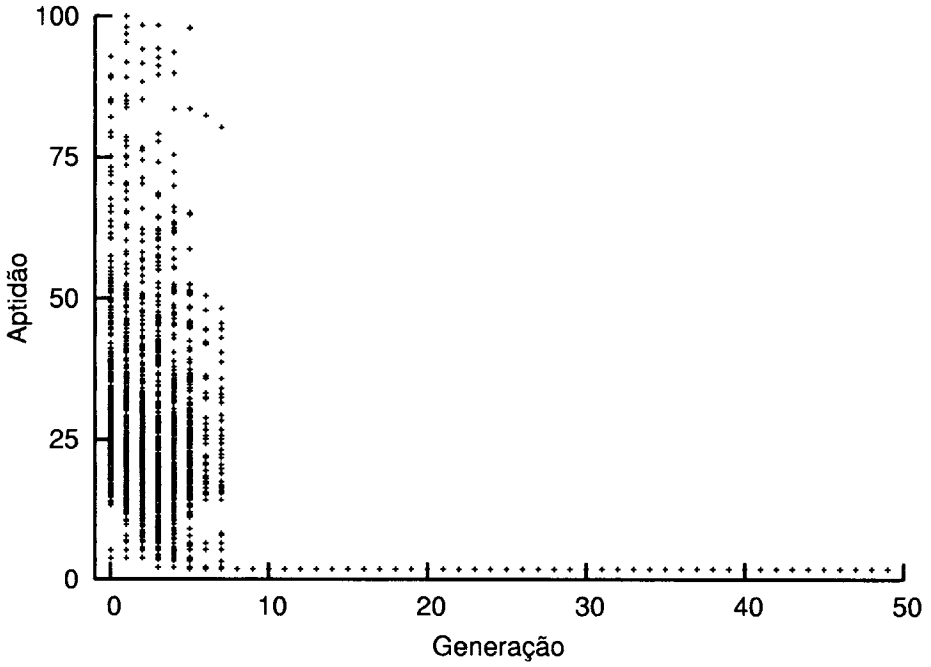


Figura 5.13: Distribuição populacional dos cromossomos gerados pela PAB sem mutação durante uma execução do segundo problema de regressão simbólica.

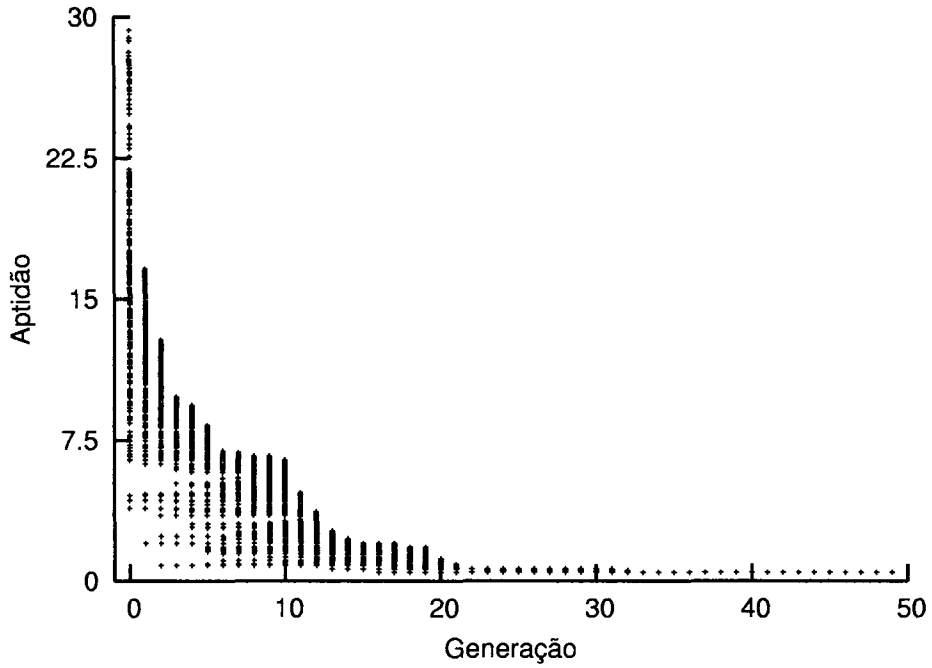


Figura 5.14: Distribuição populacional dos cromossomos gerados pela PG durante uma execução do segundo problema de regressão simbólica.

Como pode ser observado na figura 5.11, a mutação proposta cumpre a função de manter a diversidade populacional. A mutação tradicional também promove diversidade, mas de forma mais limitada, como ilustrado pela figura 5.12. Os efeitos da ausência de mutação podem ser observados na figura 5.12. A rápida convergência da PAB reduz rapidamente a diversidade populacional. Na setima geração a população já se tornou uniforme. A figura 5.14 motra que a PG também apresenta uma baixa diversidade populacional nesse experimento. No entanto, a convergencia mais lenta da PG faz com que a população mantenha uma certa diversidade até a vigésima geração. Se observarmos a figura 5.6 veremos que é exatamente até essa geração que a maioria das soluções são encontradas.

Ao analizarmos a diversidade populacional da PAB com a mutação proposta, a diversidade da PG e os resultados dos dois algoritmos com relação a freqüência de acertos acumulados no segundo problema de regressão simbólica, concluimos que o baixo desempenho da PAB nesse problema se deve principalmente ao fato da função de aptidão escolhida, soma dos erros acumulados, não ser capaz de atribuir valores adequados de forma

a promover a convergência para o ótimo global. Por exemplo, a função $x + x * x + x$, estruturalmente mais semelhante à função objetivo, recebe valor de aptidão de 7.77135, enquanto que a função $(exp((log(log(exp(exp(x))))*(cos(x)+cos(x))))*x)$ recebe 3.78765. Somado a isso, o período dos valores fornecidos como dados de treinamento, entre -1 e 1, quando aplicados as operações permitidas para esse problema, constumam retornar valores próximos a 0. O que dificulta a diferenciação das equações que levarão a convergência para um ótimo global daquelas que levam a um ótimo local.

A partir desses fatos, concluimos que o segundo problema de regressão simbólica, é um caso particular onde a PAB não apresenta bons resultados.

CAPÍTULO 6

CONCLUSÃO

As limitações encontradas nos Algoritmos Evolucionários, como por exemplo, grande quantidade de parâmetros e problema de acoplamento, fomentaram o desenvolvimento de algoritmos que mantêm uma representação explicita da população através de um modelo de probabilidade. Algoritmos que seguem esse princípio são chamados de Algoritmos de Estimação de Probabilidade (EDAs).

Neste trabalho apresentamos uma nova técnica, chamada Programação Automática Bayesiana (PAB), que usa uma representação linear de uma árvore de sintaxe concreta para estender o modelo de aprendizado através de redes Bayesianas introduzida no BOA, para o contexto da programação automática. Em adição, para tentar evitar a estagnação da evolução em um ótimo local, propomos um operador de mutação cuja operação é desencadeada pelo grau de similaridade entre o cromossomo e ser mutado e uma amostra da população.

A PAB apresenta algumas limitações, tal como a restrição quanto ao uso de cromossomos longos, dada a ordem de complexidade do algoritmos de aprendizado da estrutura da rede, que o torna mais lento que a PG.

Há ainda a tendência de gerar cromossomos muito pequenos ou muito grandes por causa da ditribuição de probabilidade irregular que os métodos de geração de cromossomos derivados de gramática sofrem.

No entanto, ele se mostrou capaz de resolver dois dos três problemas propostos. Nesses problemas, ele mostrou desempenho superior ao obtido pela PG clássica, com relação com o número de soluções ótimas encontradas e com a relação velocidade de convergência para o ótimo global. Em um dos problemas de regressão linear o seu desempenho foi inferior ao da PG, mas esse resultado pode ser atribuído a configurações particulares ao experimento proposto.

Os bons resultados com ambos os problemas estudados incentiva a sua aplicação em novos domínios de problemas.

A informação do indivíduo descrita pela rede Bayesiana pode ser melhor explorada agregando-se novas informações, como a estrutura da árvore de sintaxe concreta do indivíduo.

Para tornar o sistema mais elegante, seria interessante introduzir um método que evitasse a produção de cromossomos com tamanho muito diferente da média do tamanho dos indivíduos usados para aprender a rede Bayesiana. Isso reduziria o número de cromossomos descartados. Uma proposta para solucionar esse problema seria usar um gerador probabilístico de vetores como o proposto em [10].

A ordenação dos genes, quando transpostos da árvore de sintaxe concreta para o representação por vetor, também é algo a ser explorado, pois trabalhos [20] mostram que esse é um fator importante para o desempenho do K2.

A mutação também pode ser explorada. Ela poderia ser feita nas TPC, como normalmente ocorre nos sistemas EDAs, ao invés de no cromossomo. Outra proposta para o método atual seria considerar diferentes parâmetros de comparação de semelhança entre um cromossomo e a população.

BIBLIOGRAFIA

[1] S. Baluja e S. Davies. Using optimal dependency trees for combinatorial optimization: Learning the structure of search space. Relatório Técnico CMU-CS-97-107, Carnegie Mellon University, 1997.

[2] Shumeet Baluja. Population-Based Incremental Learning: A method for integrating genetic search based function optimization and competitive learning. Relatório Técnico CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.

[3] Shumeet Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Relatório Técnico CMU-CS-95-193, Carnegie Mellon University, Pittsburgh, PA, 1995.

[4] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, e Frank D. Francone. *Genetic programming: An Introduction.* Morgan Kaufmann, 1 edition, 1998.

[5] Jorge M. Barreto. *Inteligência Artificial. No Limiar do Século XXI.* ρρρ Edições, 1997.

[6] G. F. Cooper e E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–345, 1992.

[7] C. Darwin. On the origin of species by means of natural selection or the preservation of favored races in the struggle for life, 1859. Murray, London, UK.

[8] J.S. De Bonet, C.L. Isbell, e P Viola. MIMIC: Finding Optima by Estimating Probability Densities. *Advances in Neural Information Processing Systems*, volume 9, 1997.

[9] Thomas Dean, James Allen, e Yiannis Aloimonos. *Artificial Intelligence: Theory and Practice.* The Benjamin / Cumming Publishing Company, 1 edition, 1995.

[10] P. Duchon, P. Flajolet, G. Louchard, e G. Schaeffer. Random sampling from boltzmann principles. *Automata, Languages and Programming*, páginas 501–513. Springer Verlag, 2002.

[11] Frederic Gruau et. al. *Advances in Genetic Programming*, volume 2, capítulo On using syntactic constraints with genetic programming, páginas 377–394. MIT Press, 1996.

[12] Langdon et al., editor. *Proceedings of the Genetic and Evolutionary Computation Conference. (GECCO).* Morgan Kaufmann, 2002.

[13] GNU. Gnu scientific library. http://www.gnu.org/software/gsl/.

[14] G Harik. Linkage learning via probabilistic modeling in the EcGA. Relatório Técnico 99010, Illinois Genetic Algorithm Laboratory, University of Illinois, 1999.

[15] G. R. Harik e D. E. Goldberg. Learning linkage. *Foundations of Genetic Algorithms 4*, páginas 247–262, 1996.

[16] G. R. Harik, F.G. Lobo, e D.E Goldberg. The compact genetic algorithm. *IEEE Conference on Evolutionary Computation*, volume 3, páginas 287–297, 1999.

[17] D. Heckerman, D. Geiger, e M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. Relatório Técnico MSR-TR-94-09, Microsoft Research, Redmond, WA, 1994.

[18] David Heckerman. A tutorial on learning with bayesian networks. Relatório Técnico MSR-TR-95-06, Microsoft Corporation, Redmond, WA, 1995.

[19] John H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 2 edition, 1992.

[20] E. R. Hruschka e N. F. F. Ebecken. Variable ordering for bayesian networks learning from data. Apresentado na International Conference on Intelligent Agents, Web Technologies and Internet Commerce - IAWTIC'2003, 2003.

[21] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[22] P. Krause. Learning probabilistic networks. *The Knowledge Engineering Review*, 13(4):321–351, 1998.

[23] Paul Topon Kumar e Hitoshi Iba. Linear and combinatorial optimizations by estimation of distribution algorithms. *9th MPS symposium on Evolutionary Computation*, páginas 99–106, IPSJ, Japan, 2002.

[24] P. Larrañaga, R. Etxeberria, J.A. Lozano, B. Sierra, I. Inza, e J.M. Peña. A review of the cooperation between evolutionary computation and probabilistic graphical models. *Second Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99. Special Session on Distributions and Evolutionary Computation*, páginas 314–324, 1999.

[25] P. Lárrañaga, R. Etxeberria, J. A. Lozano, e J. M. Peña. Optimization by learning and simulation of Bayesian and Gaussian networks. Relatório Técnico EHU-KZAA-IK-4/99, University of the Basque Country, 1999.

[26] Bruce McKenzie. Generating strings at random from a context free grammar. Relatório Técnico TR-COSC 10/97, University of Canterbury, New Zealand, 1997.

[27] H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, volume 3, páginas 303–346, 1998.

[28] H. Mühlenbein e T. Mahnig. The factorized distribution algorithm for additively decomposed functions. *Proceedings of the 1999 Congress on Evolutionary Computation*, páginas 752–759. IEEE press, 1999.

[29] Michael O'Neill e Conor Ryan. Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–357, 2001.

[30] M. Pelikan, D. E. Goldberg, e E. Cantú-Paz. BOA: The bayesian optimization algorithm. *GECCO '99: Proceedings of Genetic and Evolutionary Computation Conference*, volume 1, páginas 525–532, 1999.

[31] M. Pelikan e H. MÄuhlenbein. The bivariate marginal distribution algorithm. *Advances in Soft Computing-Engineering Design and Manufacturing*, páginas 521–535, 1999.

[32] Alain Ratle e Michele Sebag. Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, páginas 211–220, 2000.

[33] Alain Ratle e Michele Sebag. Avoiding the bloat with probabilistic grammar-guided genetic programming. *Artificial Evolution 5th International Conference,Evolution Artificielle, EA 2001*, volume 2310, páginas 255–266, 2001.

[34] Genetic Algorithms Research e Applications Group *GARAGe*. Lilgp. http://garage.cps.msu.edu/software/lil-gp/lilgp-index.html.

[35] Elizabeth M. Rudnick, Janak H. Patel, Gary S. Greenstein, e Thomas M. Niermann. Sequential circuit test generation in a genetic algorithm framework. *Design Automation Conference*, páginas 698–704, 1994.

[36] Stuart Russell e Peter Norvig. *Artificial Intelligence, A Modern Approach*. Prentice Hall, 2 edition, 2003.

[37] Rafal Salustowicz e Juergen Schmidhuber. Evolving structured programs with hierarchical instructions and skip nodes. *Proceedings of the* 15^{th} *International Conference on Machine Learning*, páginas 488–496. Morgan Kaufmann, 1998.

[38] G. Syswerda. Uniform crossover in genetic algorithms. *In 3th International Conference on Genetic Algorithms, ICGA'89*, páginas 2–9, 1989.

[39] Rasmus K. Ursem. Diversity-guided evolutionary algorithms. *Proceedings of Parallel Problem Solving from Nature VII (PPSN-2002)*, páginas 462–471. Springer Verlag, 2002.

[40] P. A. Whigham. Grammatically-based genetic programming. *Workshop on Genetic Programming: From Theory to Real-World Applications*, páginas 33–41, 1995.

[41] Q. Zhanga, J. Sun, E. Tsang, e J. Ford. Hybrid estimation of distribution algorithm for global optimsation. *Engineering Computations*, 21(1):91–107, 2004.

GLOSSÁRIO

AE: Algoritmo Evolucionário.

AG: Algoritmo Genético.

BMDA: Algoritmo de Distribuição Marginal Bidimencional (Bivariate Marginal Distribution Algorithm).

BOA Algoritmo de Otimização Bayesiana (Bayesian Optimization Algorithm).

BSC: Cruzamento Simulado Baseado em Bit (Bit-Based Simulated Crossover).

CE: Algoritmo de Estimação de Rede Bayesiana (Estimation of Bayesian Network Algorithm).

CE: Computação Evolucionária.

cGA: Algoritmo Genético Compacto (Compact Genetic Algorithm).

COMIT: Otimizadores Combinantes com Árvores de Informação Mutua (Combining Optimizers with Mutual Information Trees).

EcGA: Algoritmo Genético Compacto Extendido (Extended compact Genetic Algorithm).

EDA: Algoritmo de Estimação de Distribuição (Estimation of Distribution Algorithm).

FDA: Algoritmo de Distribuição Fatorizada (Factorized Distribution Algorithm).

GE: Evolução Gramatical (Grammatical Evolution).

GLC: Gramática Livre de Contexto.

MIMIC: Agrupamento de Entradas por Maximização de Informações Mutua (Mutual Infromation Maximization for Input Clustering).

PAB: Programação Automática Bayesiana.

PBIL: Aprendizado Incremental Baseado em População (Population-Based Incremental Learning).

PG: Programação Genética.

PIPE: Evolução de Programa Incremental Probabilístico (Probabilistic Incremental Program Evolution).

PPT: Árvore Protótipo Probabilística (Probabilistic Prototype Tree).

SG-GP: Programação Genética Estocástica baseada em Gramáticas (Estocastic Grammar based Genetic Programming).

TPC: Tabela de Probabilidade Condicional.

UMDA: Algoritmo de Distribuição Marginal Unidimencional (Univariate Marginal Distribution Algorithm).